Interface

2004
5
Contents

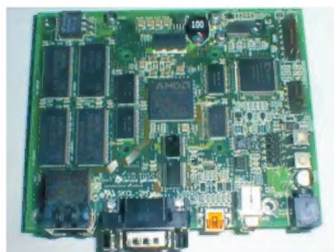


［表紙デザイン：㈱プランニング・ロケッツ］

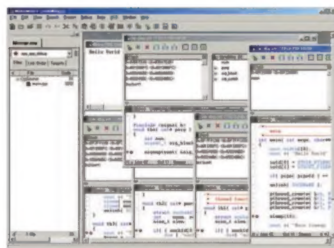
特集
社会を支えるインフラストラクチャ
組み込みシステムの世界へようこそ!
Cover Story
Welcome to the world of embedded systems! 35

第1章 遠くて近い，「組み込み」の世界
組み込みシステムとは何か 36
Chapter 1 What is an embedded system?
猪飼 國夫(Kunio Yikai)

第2章 地の底から宇宙の果てまでを支える
社会のインフラ——組み込みOS 47
Chapter 2 A social infrastructure —— The embedded OS
藤倉 俊幸(Toshiyuki Fujikura)



第3章 動かない！を動かすための
実例で学ぶハードウェアのデバッグ 55
Chapter 3 Hardware debugging learned through an example
川本 泰久(Yasuhisa Kawamoto)



第4章 異なった環境で動作するソフトウェアをPCで開発する
組み込み向けクロス開発環境の構築 66
Chapter 4 Construction of a cross development environment for embedded systems
中村 憲一(Kenichi Nakamura)

第5章 CodeWarriorを使用した組み込み開発
統合開発環境を用いた組み込み開発の事例 76
Chapter 5 Example of an embedded system development using IDE
深瀬 茂寛(Shigehiro Fukase)

特別付録 **ISAバス＆Cバス活用ハンドブック**
A separate booklet appended to a magazine
ISA Bus & C Bus handbook

Interface

2004 5

Contents

話題のテクノロジ解説

ROM-LinuxChipで実現する
組み込みLinuxのROM化の実例 87
An example of ROM-ing of Embedded Linux
中島 信行(Nobuyuki Nakashima)

TOPPERSで学ぶRTOS技術(第6回)
サービス・コールの概要・その3 98
Summary of the service call (Part 3)
岸田 昌巳(Masami Kishida)

オープン・ソース・ソフトウェアDSP Gatewayを用いた
Linuxで使うOMAP DSP部のソフトウェア開発環境 156
Software development environment of OMAP DSP used in Linux
森 英悟/小林 俊浩/高橋 清隆
(Eigo Mori/Toshihiro Kobayashi/Kiyotaka Takahashi)

米国マクデータ社ジョン・ケリー氏に聞く
iSCSI技術のストレージ製品への応用 165
Application of iSCSI technology in storage products
北村 俊之(Toshiyuki Kitamura)

ショウレポート&コラム

企業のためのITソリューション展
NET&COM 2004 13
北村 俊之(Toshiyuki Kitamura)

ハッカーの常識的見聞録
いよいよ出てくる64ビット対応Prescotteに期待しよう 17
Look forward to finally appearing Prescotte for 64bit
広畑 由紀夫(Yukio Hirohata)

移り気な情報工学
性善説と性悪説で考えるRFID 19
RFID considered in both well-natured and ill-natured theories
山本 強(Tsuyoshi Yamamoto)

シニアエンジニアの技術草子(参拾八之段)
時は移れど 180
Although the time has passed
旭 征佑 (Shousuke Asahi)

Engineering Life in Silicon Valley
フリー・エンジニアという仕事(第二部) 182
The work of free engineers (Part 2)
H.Tony Chin

IPパケットの隙間から
強引なセールスとネット社会の成熟 190
Aggressive sales and maturity of the net society
祐安 重夫(Shigeo Sukeyasu)

一般解説&連載

TMS320C6713搭載DSPスタータ・キットを使ったC++によるDSPオブジェクト指向プログラミング(第4回)
FFTクラスの作成 114
Making of an FFT class
三上 直樹(Naoki Mikami)

開発技術者のためのアセンブラ入門(第25回)
アセンブラMASMおよびgasでのSSE/SSE2命令の使いかた 127
How to use SSE/SSE2 instructions in Assembler MASM and gas
大貫 広幸(Hiroyuki Oonuki)

プログラミングの要(第11回)
リスト構造とその応用 133
List structure and its application
宮坂 電人(Dento Miyasaka)

組み込みプログラミング・ノウハウ入門(第14回)
アクティブ・オブジェクト・モデリングのこころ――順序集合分割法(その2) 139
The notion of active object modeling ―― Ordered set division (Part 2)
藤倉 俊幸(Toshiyuki Fujikura)

やり直しのための信号数学(第23回)
DCTによる信号処理応用(その2) 146
Applications of the signal operation using DCT (Part 2)
三谷 政昭(Masaaki Mitani)

フリーソフトウェア徹底活用講座(第15回)
GCCにおけるマルチスレッドへの対応 170
Measures for multi-threads in GCC
岸 哲夫(Tetsuo Kishi)

情報のページ

Show & News Digest	15
NEW PRODUCTS	184
海外・国内イベント/セミナー情報	191
読者の広場	192
次号予告	194

連載「SDIOカード開発入門」と『VxWORKS』を使ったRTOS技術の基礎と応用」は，お休みさせていただきます．

企業のための IT ソリューション展

NET&COM 2004

北村 俊之

「変革への扉を押し開く，新技術と新サービスがここに集結！」をテーマに「NET&COM 2004」が 2 月 4 日(水)～6 日(金)の 3 日間，幕張メッセで開催された．主催は日経 BP 社．1993 年の「Open System Expo」から数えて，今年で 12 回目の開催となる．ここ数年で驚異的な飛躍を見せているネットワーク技術は，社会のしくみや行動に急激な変化を与えている．とくにビジネスにおいては，高速なネットワークを基盤とした企業情報システムをいかに戦略的に構築できるかで，その企業の生き残りが決まってくるとさえいわれている．また，今後，グリッド・コンピューティングや自律コンピューティングなど次世代コンピューティングの技術や概念の登場にあたって，これらの技術をいち早く活用することがより重要な課題である．

本展示会では，展示会全体を「ネットワーク」，「システム構築/運用」，「セキュリティ」，「Web ソリューション」，「CT/CRM」の五つの展示ゾーンに分け，さらにこれらの展示ゾーンの中に「ネットワーク/システム運用管理」，「VoIP」，「ビジュアルコミュニケーション」，「US Technologies」，「ストレージ」，「GIS」，「バイオメトリクス」など個別のテーマに焦点を当てた 12 のパビリオンを設置することで，話題の技術，ソリューション情報の提供が行われていた．また，展示会と併催された「NET&COM 2004 FORUM」では，IT ベンダによるスペシャル・セッションや特別講演のほか，専門セミナとして全 3 トラック，17 セッションや 50 を超えるセミナが開催されていた．最終的な来場者数は，66,396 人だった．

● セキュリティゾーン

セキュリティゾーンでは，シマンテック，シスコシステムズ，ネクサンティス，キヤノンシステムソリューションズなど，多くのセキュリティ・ソリューション・ベンダが出展しており，会期直前に世界的に蔓延したウィルス「MyDoom」の影響もあり，各社のブースは大盛況だった．シマンテック(写真 1)は，もはや単一のソリューションだけでは十分な対策とはいえないことをアピールし，複合型の脅威から企業の資産やネットワークを保護するための多彩なトータル・セキュリティ・ソリューション製品の展示デモを行っていた．そうした中で「Symantec Gateway Security 5420」などのアプライアンス製品や，「Symantec Client Security」などのソフトウェア・ソリューションが来場者の注目を集めていた．

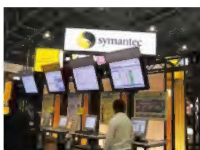


写真 1　シマンテックのブース

シスコシステムズは，2003 年 12 月に発表された，端末でのセキュリティ・レベルを確認し，企業ポリシに準拠していない端末のアクセスをルータ，スイッチ，セキュリティ製品で制御する NAC(Network Admission Control)を中心とした展示を行っていた．同社としては，運用担当者の負荷軽減などを考慮した，システムとしてのセキュリティ構築の提案を行っていくとのことだった．また，無線 IP 電話「7920」(写真 2)などの展示も行われており，こちらも来場者の関心を集めていた．


写真 2　シスコの無線 IP 電話 7920

ヒューマンテクノロジーズは，低価格な指紋照合システム「U.are.U 4000 シリーズ」(写真 3)および同製品を活用した勤怠管理 ASP サービス「King To Time」の展示デモを行っていた．「U.are.U 4000 シリーズ」は光学式指紋センサでありながら，一般的な半導体式なみのサイズを実現しており，精度，価格面においてもコスト・パフォーマンスの高い製品となっている．


写真 3　ヒューマンテクノロジーズの U.are.U 4000 シリーズ

● システム構築/運用ゾーン

ポートウェルジャパンは，DVR 製品，さまざまなネットワーク機器に対応可能なハードウェア・システムを多数展示していた．工業用マザーボードを採用し，安定運用を実現した監視システム「PVS-1121」や，家庭および小規模オフィスでの使用を目的とした監視システム「PVS-1110」などの製品が注目されているという．「PVS-1121」では，100Base-T Ethernet ポート装備，Windows IE ブラウザでの遠隔操作，録音/再生などの同時操作，120fps で表示，録画のリアルタイム映像が可能，MPEG4 をサポートなどの特徴をもつとのことだった．

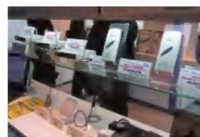
● ネットワークゾーン

東陽テクニカは，無線 LAN，VoIP をテーマとした展示を行っていた．中でも無線 LAN モニタ，監視システムである「AirMagnet」(写真 4)の人気が高いとのことであった．同製品は，ワイヤレス・ネットワークの測定，診断を行うツール．適切なワイヤレス LAN を構築するために必要な情報をさまざまな角度から収集し，接続性からセキュリティのチェックなど幅広い情報の測定，解析を可能としている．

セイコーインスツルメンツは，RAS/コミュニケーション・サーバ「NS シリーズ」を中心に展示を行っていた．ATM と Ethernet の連携機能を実現し，広域 Ethernet サービスのアクセス回線を効率よく収容するエッジ・スイッチ「EXAtrax」の，冗長性をさらに向上させた新モデルを展示していた．また，ATM-Ethernet コンバータまたは ATM ルータで動作可能な「BlueBrick」も人気の製品だという．こちらは，優先/帯域制御などの QoS 機能を備え，VoIP アプリケーションに適するとのこと．


写真 4　東陽テクニカの AirMagnet Handheld

コンテックは，IEEE802.11a/g 準拠 54Mbps 無線 LAN「FLEXLAN DS540 シリーズ」(写真 5)，IEEE802.1X 認証サーバ・ユニット，ネットワーク管理ソフトウェア，無線 LAN 対応授業支援ソフトウェアなどの製品を幅広く展示していた．また，通信コストの削減を実現する新製品「無線 LAN ビル間通信ユニット」も来場者の関心が高いとのことだった．「FLEXLAN DS540 シリーズ」は，5GHz 帯 4 チャネル，2.4GHz 帯 3 チャネルの合計 7 チャネルから，2 チャネルを同時に提供できるため，帯域が実質 2 倍となり，帯域不足を解消した高速無線 LAN の実現が可能になるという．


写真 5　コンテックの FLEXLAN DS540 シリーズ

ソニーマーケティングは，小型，軽量，高画質でスピーディな意思決定を支援するビデオ会議システム「PCS-1」(写真 6)，コマーシャルから VOD まで，多様な用途に対応するコンパクトなシングル・チャネル・サーバ「NSP-100」などの展示を行っていた．


写真 6　ソニーマーケティングのビデオ会議システム PCS-1

Emblix NPO法人設立記念セミナー/IPA未踏事業 中島プロジェクト成果報告会

■ 日時：2004年2月18日(水)
■ 場所：虎ノ門パストラル(東京都港区)

産学共同によるEmbedded Linuxの組み込み業界への普及および浸透を目的として任意団体として設立されたEmblixが，2003年10月7日にNPO法人化されたことを記念してセミナを開催した．また，これと同時にEmblix会長の中島 達夫氏がプロジェクト・マネージャを勤める，IPA未踏事業中島プロジェクト 成果報告会も同時開催された．

経済産業省 情報処理振興課長補佐 久米 孝氏による講演「組込みソフトウェアの活性化に関する政府の施策」では，経産省としては現在の組み込み産業に関して，市場規模・価格総額などを含めた現状を把握できていないということを挙げ，組み込み産業の実態調査を開始したことを発表した．また，組み込みソフトウェアのスキル標準を策定し，国内の組み込み技術全体を底上げすることが必要だと語った．

続くIPA成果報告会では，未踏プロジェクト参加者がプレゼンテーションを行った．早稲田大学の菅谷みどり氏による「ロバストな組み込み向けオペレーティング・システム」は，Linuxにフェア・スケジューリングによるクラス別スケジューラを実装したもの．また，LinuxとUML(ユーザ・モードLinux)を組み合わせることにより，資源予約機能を実現した．

続く佐藤 嘉則氏の「uClinuxのH8/300アーキテクチャ移植」は，MMUをもたないCPUであるH8/300にLinuxを移植したもの．Linux 2.5.xへの追従と，toolchainの改良によるメモリ使用量の削減を主眼において開発を行ったとのこと．とくに後者は，GCC 3.4 for H8がPIC(Position Independent Code)環境に対応したことをうけ，Linuxカーネルとtoolchain側もこれに対応することにより，PICバイナリを実行できるようにしたとのことだ．

ほかには，豊橋技術科学大学の本田 晋也氏による「ソフトウエア・ハードウエア間インタフェース生成ツールの開発」，メタウエアリサーチ(有)の丸一 威雄氏による「位置情報を扱うユビキタスサーバの開発とGISでの利用」，慶応義塾大学の岩井 将行氏による「プライバシを考慮する分散型位置管理機構Tachyonの開発」などのプレゼンテーションが行われた．


経済産業省 情報処理振興課長佐
久米 孝氏

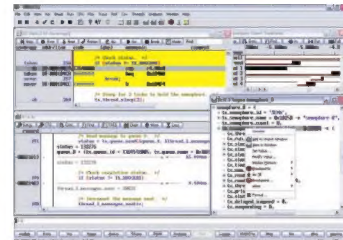
日本ローターバッハ，PowerToolsを発売

■ URL：http://www.lauterbach.com/

日本ローターバッハ(株)は，JTAGデバッガTrace32 PowerToolsを発売した．JTAGエミュレータを中心にデバッガおよびパフォーマンス解析ツールなどのソフトウェアをセットにしたもので，マルチコア・デバッグも可能．対応CPUはARM 7/9/10/11(開発中)，OMAP，PowerPC，SH，H8，V850など．対応OSはSymbian，VxWORKS，QNX，Linux，Windows CE，OSE，ThreadXなど．

とくにLinuxは，MMUを用いて物理・論理アドレス変換を行うためデバッグが難しかったが，デバッガ側でアドレス変換テーブルを認識するため，通常の論理アドレスでデバッグが可能になる．カーネルでもユーザ・アプリケーションでも同様にデバッグが行えるほか，この機能はカーネルを無改造で使うことができることが特徴である．

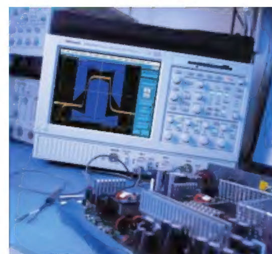

PowerTools

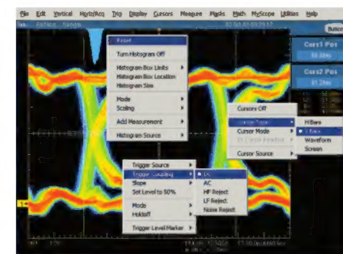

PowerToolsの画面

日本テクトロニクス，使いやすさに注目したミッド・レンジのオシロスコープ「TDS5000Bシリーズ」を発売

■ 日時：2004年2月26日(木)
■ 場所：日本テクトロニクス本社(東京都港区)

日本テクトロニクスは，ミッド・レンジ・クラスのオシロスコープ「TDS5000Bシリーズ」を発売した．同社によると，ミッドレンジ・クラスのオシロスコープに一番求められていることは，「使いやすいこと」であるという．そのため，本製品は使いやすさのための二つの機能を備えている．一つは，画面をマウスで右クリックすると関連した項目がドロップ・ダウン・メニューとして表示される「右クリック」機能であり，もう一つは「MyScope」と呼ぶ機能である．「MyScope」機能は，ユーザがトリガ機能の選択や帯域制限などについて設定するためのメニューを，マウスを使って簡単に設定できる機能である．この機能は，オートセット・ボタン以来の使いやすい機能だという．


TDS5104B


右クリック機能

ハッカーの常識的見聞録

広畑 由紀夫



今月の常識

いよいよ出てくる64ビット対応 Prescotteに期待しよう

☆ 今回はIDF(Intel Developer Forum)Spring 2004において，ついに発表された「Intel 64Bit Extension Technology」について簡単に説明します．

2004年2月現在，発売が開始されている，Pentium4 Prescotteコアでは，SSE3やL2キャッシュ容量，パイプラインなどに関して多くの拡張がなされていますが，プロセッサ自身の機能としては従来のハイパ・スレッド対応のPentium4をほぼ継承しているといえます．一部のラインナップでは，ローエンド・デスクトップ向けにハイパ・スレッドなしの低価格品も提供しているとのことなので，Pentium4 Extreme Editionとは用途ごとに入れ換わっていくことになるでしょう．

● 64ビット拡張によるメリット

さて，64ビット拡張機能が付くことで何が変わるのでしょうか．

現状では，64ビット演算などによる高速化によって改善されるのは，4Gバイトを超えるメモリ・アクセスや，ファイル操作関連です．とくに，NTFSフォーマットによる大容量HDDへの総合的なパフォーマンスの向上が期待できます．

● 家庭用64ビット・プラットホームは必要か？

Intel社以外の64ビットに対応したCPUでは，メモリ・アクセスの高速化が強調されていますが，物理メモリで4Gバイトを超えるメモリ容量を搭載できる家庭用コンピュータは事実上存在しません．また，64ビットに対応したWindowsの出荷待ちの状態であることも確かです．そのため，家庭向けの64ビット対応Windowsのプラットホームとしては，Windows XP SP2以降で，なおかつ物理的に8Gバイト以上の物理メモリが搭載できる本体で，少なくとも2Gバイト以上のメモリを実装したPCが必要になってくるようにならないと，64ビット拡張機能を有効に使えるようにはならないでしょう．

現在の32ビット・プラットホーム用のソフトウェアがそのまま動作するとはいえ，64ビットの拡張機能による速度の向上が大して期待できないので，少なくとも最低で4Gバイトのメモリが必要なアプリケーションなどを動作させる環境が必要になるまで待ったほうが良さそうです．

● IntelとAMDの64ビット拡張

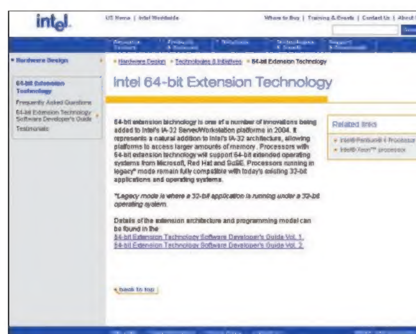
Pentium4における64ビット機能拡張コードにかかわるライセンスなどの問題については，AMD社との間で調整がついているらしく，バイナリ・コードの実行環境においては問題ないとのことです．そのため，64ビット拡張機能に対応したWindowsの出荷後，AMDの製品とともにPentium4 Prescotte版64ビット機能拡張が利用できるようになると思われます．

● 開発環境

Windows向けのソフトウェア開発キットとしては，将来に出荷されるだろうVisual Studio.NET，もしくは現行バージョンへの追加オプションというかたちで対応できるようになるのではないかと考えられます．一般的なアプリケーションをVisual Studio.NETなどの統合環境で作成する場合には，コンパイラ・オプションやリンク・ライブラリに64ビット拡張機能対応版を使用するだけになることでしょう．

● 今後のPentium4に寄せる期待

今後，64ビット拡張機能だけではなく，デュアルコア/マルチコア技術といったサーバ向けや家庭向けの機能が実装されると考えられます．今回実装されたSSE3やL2キャッシュの増量は，家庭用PCなどにおいてマルチメディア化を促進するアプリケーションやデバイス・ドライバなどの環境が整ってくるにつれて実際に役立つようになってくることでしょう．また，ハイエンド・ゲーム向けに，Extreme Editionのような大容量のL3キャッシュが必要になるといった需要が発生すれば，たとえ高価なものになるとしても，生産される可能性はあると考えています．筆者は，すでにPentium4 Extreme Editionと，2004年2月出荷版のPrescotteでゲームなどを楽しんでいますが，どちらも一長一短という感じです．将来的にハイエンド・ゲーム向けとして，さらにL3キャッシュや，L2キャッシュの増量を含めた機能強化を行ってほしいと期待しています．



ひろはた・ゆきお　OpenLab.

■参考URL
(1) IDF Spring 2004公式サイト
http://www.intel.co.jp/jp/idf/
(2) Intel 64-bit Extension Technology(英語)
http://www.intel.com/technology/64bitextensions/

移り気な情報工学

性善説と性悪説で考えるRFID

山本 強

RFID，いわゆる無線タグが今注目されている．ゴマ粒大のICチップがバーコードを置き換えるとか，レジ清算がカゴごと一瞬でできるとか，食品の賞味期限を冷蔵庫が認識するとか，ともかくいろいろな話題が経済誌をにぎわせている．その先にはスマート・ダストというセンサ内蔵無線タグのクラスも控えている．最近ネタ不足気味のIT業界としては，久々の長期有望分野の出現だともいえる．しかもRFIDはすでに動いているシステムであり，コストの問題さえ解決できれば今すぐにでも使える技術なのである．ドッグ・イヤーとかマウス・イヤーで動いているといわれるIT業界なのだから，それほど見えている技術ならば，話題だけでなくもっと広く使われて良いのではないかと思うわけである．

今回はRFIDにまつわる二つの話題を性善説と性悪説という視点から検証してみたい．

レジレス・コンビニのリアリティ

買い物かごをゲートに通すだけで自動的に購入金額が計算され，電子マネーで決済できるレジレス・コンビニというコンセプトがある．RFIDの一般向け応用としては，これが一番わかりやすい例である．タグのコストは将来1円以下にまで下がるというのだから，商品価格が100円以上ならば現実性はある．技術的には読み取り精度も限りなく上がるだろう．システム的にもバーコードをRFIDに置き換えるだけだから，RFID付き商品だけを取り扱うコンビニならば，今でも実現可能である．

問題はこのコンビニの運用形態にある．このレジレス・コンビニ，全商品がRFID対応で，かつ客がすべて電子マネー対応になっていることが前提になる．電子マネーを使わない客を認めたとたんに，レジ係が必要になり，レジがあるならタグなし商品も扱えるということになって，どんどん普通のコンビニと化してしまう．

少し積極的に考え，RFID商品と電子マネーだけを取り扱うレジレス・レーンを作るという考えもある．しかし，そのレジレス・レーンではRFIDなしの商品はリーダから見えず，意図しようとしまいとチェックされずに通過できてしまうことになるから，やっぱり全商品RFID付きにせざるを得ない．結局，扱い商品の少ない不便なコンビニとなってしまう予感がする．コンビニに限らず，小売業では基本的に顧客を性悪説で考えている．

RFID対応冷蔵庫に見るトレーサビリティの夢

RFID読み取り機能付き冷蔵庫という夢もわかりやすい．RFIDに商品情報を定義しておき，冷蔵庫に商品を入れると自動認識され，冷蔵庫が賞味期限などを教えてくれるというものである．この夢が画期的なのは，商品情報を1個単位で提供するという点である．情報科学的に言えば，既存のバーコードは商品に共通な基本クラス情報だけを提供するが，RFIDは個々の商品のインスタンス情報まで提供するということである．技術的には，商品1個1個の情報を提供するデータベースの維持管理コストと，情報流通コストが問題になるのは明白である．

とりあえず，それはe-Japan戦略が保証する世界最強のITインフラが何とかしてくれるとしよう．それでもこのシステム，IDとデータベース上の商品情報が一致していることをどうやって保証するかという問題が残るのである．RFIDが製品と構造的に不可分であるならば，IDと商品が一致しているとみなせるが，生鮮食料品は流通で次第に小分けされていくので，分割のたびにIDの付け替えが行われる．このモデルには生産者も流通業者もみな正直で商品情報を偽らないという性善説の仮定が入っている．逆説的になるのだが，流通側にRFIDを入れてまで生産地や賞味期限を正しく伝えたいという“善意”があるのなら，トレーサビリティが問題になることはなかったのである．

例に取り上げた二つのシステムが機能するためには利用者もサービス提供者も情報を正しく提供し，悪意はないという仮定が入っている．また，RFIDは非接触読み出しであり，商品購入後も生きているIDなので，その人が何を持っているかが外部から見える，つまりプライバシが侵害されるという指摘もある．これもRFIDのリーダを持つ組織は悪いことはしないという性善説的な仮定に無理があるから出てくる話である．

RFIDはもともと消費者まで情報が流通するという開放型の応用が想定されていないシステムだったように思う．企業内の在庫管理や流通管理のように信頼可能な範囲での応用に向いているから，そういう用途には地道に浸透してきている．

RFIDが可能にする夢のサービスを実現するためには，人間の行動原理を含めたシステム設計にひとひねりくふうがいるように思えるのである．

やまもと・つよし　北海道大学大学院情報科学研究科
メディアネットワーク専攻



特集

社会を支えるインフラストラクチャ

組み込みシステムの世界へようこそ!



新入社員を迎える4月直前ということで，今月はフレッシャーズ特集です．

本誌でたびたび取り上げているキーワードとして「組み込みシステム」があります．組み込みシステムは表立って取り上げられることは少ないものの，家電という形で日常生活の中に幅広く溶け込んでいるばかりでなく，宇宙空間へと飛び立つような先端的な分野をもサポートする幅広い技術です．

「パソコンのようでパソコンでない」組み込みシステムは，たしかにCPUとメモリとI/Oをもった電子回路という形態をとっています．しかしパソコンと違い，可能な限り低消費電力を目指し，可能な限り小型化し，その中で可能な限り性能を追求しています．そのためにはパソコン向けとは違う，組み込み向けに特化したCPUやOSが使われます．

そんな「もう一つの世界」である組み込みシステムへ，一歩踏み出してみませんか？

1. 遠くて近い，「組み込み」の世界
組み込みシステムとは何か
猪飼 國夫

2. 地の底から宇宙の果てまでを支える
社会のインフラ――組み込みOS
藤倉 俊幸

3. 動かない!を動かすための
実例で学ぶハードウェアのデバッグ
川本 泰久

4. 異なった環境で動作するソフトウェアをPCで開発する
組み込み向けクロス開発環境の構築
中村 憲一

5. CodeWarriorを使用した組み込み開発
統合開発環境を用いた組み込み開発の事例
深瀬 茂寛

1. 遠くて近い，「組み込み」の世界

組み込みシステムとは何か

猪飼 國夫

あまり表舞台に出てこないにも関わらず，生活に密着したシステム，それが組み込みシステムである．そもそも組み込みシステムとは何なのか，特集の始めにこの「組み込み」という概念を定義し，組み込みの応用範囲を概観してみる．（編集部）

「組み込みシステム」と呼ばれるシステムは一般的なPCとは違い，x86以外のCPUが使われ，独自の基板を起こし，Windowsではない OSが搭載されることがあります．これらの理由はなぜなのかについて，おもに歴史的な変遷を元に，ごく初歩的な解説を行います．過去をふりかえる理由は，CPUが技術の発展にともなって，組み込みの対象に最適な形で進化してきたからです．

組み込みシステムとは

そもそも「組み込みシステム」とは何なのかについて，あれこれと考察してみます．

● 何を組み込むのか

組み込みシステムということばは英語のembedded systemを日本語にしたものです．苦手な英語についての講釈はコラム1を読んで理解してもらうことにして，組み込みということばの定義を決めたいと思います．

ずばり，組み込むのはコンピュータのCPU/MPU（Central/Micro Processing Unit）です．では，パソコンも「組み込み」かというと，じつはそのとおりです．でも，パソコンを組み込みシステムとして考える人はあまりいないと思いますし，ここで議論する対象でもなさそうです．

● 組み込みシステムというものは独立した概念か

では組み込みシステムとは何でしょう．組み込みに対立することばは古めかしいですが，「スタンド・アローン（それ自体で独立している）なコンピュータ」という用語が最適でしょう．スタンド・アローンでないものが組み込みシステム，という反対語での意味付けではもの足りませんが，何となく気分は理解できそうです．

しばらく前までは，「汎用キーボードなどがつながっていてコンピュータとして使えるもの，ではないもの」などという人もいましたが，じつは20年以上前でもNC工作機械の制御などには，パネルと称するディスプレイとキーボードが付いていて，PC-DOS（MS-DOS）とその上でPC-DOS用のアプリケーションが走るようになっていました．筆者の認識では，このようなものもやはり組み込みシステムだと考えます．

本章では，とりあえず汎用のパソコンとして使われることがないものは基本的に組み込みシステムであると考えることにします．そうすると，最近のゲーム機のようにキーボードと何かちょっと付けるとLinuxが走る汎用機になるようなものは，どちらなのでしょうか．

● 組み込みとパソコンとの間のグレーゾーン

高級な計測器の中にはx86とWindowsやUNIXを組み込んで，計測結果を記録するだけでなく，種々の処理を行ったり，LANで接続できるものがあります．一方，パソコンにA-D/D-A変換機能やディジタル入出力のアダプタを付けて計測器や制御器として使えるシステムもあります．両方ともまったく同じことができますが，前者は組み込みシステムといい，後者はパソコン・システムと理解されています．このように，組み込みという概念は，どちらともいえない灰色の部分が多く，一概に断定できません．

組み込み用のCPUチップを使ったものが組み込みシステム

COLUMN 1

embedded

辞書を引くと，bed：寝床，機械や道具の土台，ワーク（被加工物）が置かれる台，などという意味の名詞です．embed（em+bed）：bedに置く，台に設置する，埋め込む，というふうにbedを動詞化したものです．embedded（embed+ded）：embedの過去分詞ですから，埋め込みの，という意味になるようです．

だという考えもありますが，それはこれまでの議論の中でわかるように無意味な限定です．正直いって，筆者のようなトランジスタ時代からの古手のコンピュータ技術者にとっては，表1のようにがんばって組み込みという概念を分離すること自体が，無意味なことのようにも思われます．

しかし，それでは特集が成り立ちませんし，この号を手にした読者も惑わされたような気になると思います．そこで，CPUが組み込み用途に使われてきた歴史的な経緯をたどることで，現在の組み込みシステムの全体像を明らかにし，多種多様な組み込みの実態とその技術を，次章からの各分野での解説で把握できるようにしたいと思います．

x86と組み込み向けCPUの違い

ここでは，x86も組み込み向けCPUも，同じCPUなのではないかという意見を歴史的に検証します．

● x86の祖先はやはり組み込み用

多くの人が知っているように，x86系のいわゆるIntelチップは，図1のようにその遠い祖先は組み込み用として開発されました．そういう意味で，今は組み込み用に使われているMIPSチップ（ゲーム機のPlayStationに使われたCPUで今のPS2やPS1にもその後継バージョンが使われている）は，由緒正しいスタンド・アローンのコンピュータ用のチップでした．

じつはx86は，遠い祖先が組み込み用に作られたという過去とIntelという会社の営業方針で，この世に顔を見せたときの遺伝子をずっと体内に持っていました．ほかのアーキテクチャのCPUチップは売れなくなったということもありましたが，そのアーキテクチャは時代の変遷に合わせて大きく変化してきました．

● 最初は電卓用の組み込みチップ

ご存じかもしれませんが，4004は日本のビジコンという会社の電卓を作るためのチップとして，Intel社が受注開発で作ったものです．筆者も4004の開発が始まる前のIntel社に，ICの製造を検討してもらいに訪れたことがあります．Silicon Valleyにある Stanford大学周辺の工業団地の一区画を占めていた，小さな会社でした．

ビジコンの計画はその後つぶれてしまったので，宙に浮いた4004を組み込み用に売り出したのです．4ビットのチップではできることに限界があったので，すぐに8ビットの8008が発表されました．このチップはセイコーの電卓に組み込むチップとして採用されました．

● x86チップがコンピュータに進出した

8008は，すぐに仕様がコンピュータ的なものに発展し，有名な8080（8080A）が世に出ました．日電との間でチップ内部のマイクロ・コードの著作権紛争になった歴史的なチップです．

8080Aはコンピュータとして使える仕様だったので，多くのスタンド・アローンの超小型コンピュータが多くのベンダ（供給元）から発表されました．これらは，当時全盛を誇っていたミニ・コンピュータからの連想でマイクロ・コンピュータと銘打って発売されました．

表1　いろいろなシステムのほんの一例

コンピュータ	灰色分類	組み込みシステム
地球シミュレータ	ビル管理，工場制御	機器制御（交通・流通，生産，計測監視など）
メイン・フレーム	ルータ	自動車制御，カーナビ，家電制御
各種サーバ	PDA	個人用ゲーム機，携帯電話
パソコン		ICカード，ICタグ

● 8ビットまでのチップはだんだんと組み込み用に

8080Aを使ったゲーム機もすばやく作られました．当時，筆者は一つ1万円もした8080Aを組み込んだ計測システムを作って，都内のビルや繊維会社の製造設備の制御装置として納入しました．

Intel社はこのような用途向けに，RAMやROM，I/Oポートを一つのチップに積んだ8048，後には8051を出しました．これは非常に多くの組み込み用途の顧客を開拓し，家電製品の電子制御化に役立ちました．

日本の電機各社も同じような組み込み用の4ビットや8ビットのマイコン・チップを出し，マイコンという用語は組み込み用の多機能チップを指すようになりました．

8080AもZilog社がZ80として発展させ，ある意味では究極の8ビットCPUとなりました．Z80は8ビット・パソコンや日本独自のワープロ用だけでなく，筆者がスロット・マシンのコントローラに採用したり，初代のゲームボーイなどにも使われたため，日本ではパソコンと組み込み用の双方で非常に多く使われるチップとなりました．

日本では8ビット・パソコンが普及したため，Z80の開発環境も入手が容易となり，Z80を組み込み用途に使う人口が多かったと思われます．Z80は教育用として今でも余命を保っています．

● OSの導入で組み込みシステムからパソコンに発展した

パソコンも8ビット時代は図2のようにBASICでプログラムを組み，I/OはBASICに内蔵されたドライバで直接動かすのが一般的でした．それを大きく変えたのがDigital Research社のCP/Mという8080A/Z80用のOSでした．

OSが提供されたため，パソコン内に組み込まれたROMで提

4004 → 4040 → 8008 → 8080 → 8080A → 8085 → 8086 → 80286 → 80386 → 80486 → Pentium → Pentium II → Pentium III → Pentium 4
→ 8048 → 8051　(→ Z80)　→ Pentium-Pro　→ Celeron

図1　Intel社のチップの変遷

供されていたBASICなどのアプリケーションは記憶装置から読み込めるようになり，ゲームのプログラムを雑誌の記事を見ながら毎回手で打込んで遊ぶ，という（今では）信じられないことをする必要がなくなりました．

IBM社がそれまでメイン・フレームの端末として使っていた通信機能付きのディスプレイとキーボードのセットを改良して，Intel社の8088（8086のバス幅限定版）を採用したIBM-PC（その後IBM-PC/XT）という名のパソコンとして一般にも売り出しました．Microsoft社のROM BASICが内蔵されていた点は，日電のPC-8001と同じです．

同じ時期にIBM社が開発したフロッピ・ディスク（IBM社では商品名をDisketteと呼ぶ）に，Microsoft社のOSをPC-DOSという名前で搭載しました．x86チップが組み込み用の世界から本格的なスタンド・アローン型のコンピュータに進出したのです．現在のパソコン時代の幕開けでした．

● 組み込みに使われるCPUチップ

現在では，組み込み用に使われるCPUチップは，大きく分けて表2のように2種類あります．すなわち汎用CPUとしてパソコンなどに使われることも考慮に入れて開発されたものと，図3のようにI/Oポートなどを内蔵していて，なるべく少ないチップ数でコントローラなどの機能を実現できるようにしたものです．

原理的にはいろいろな機能を取り込んだ組み込み用のチップでも，パソコンなどスタンド・アローンなコンピュータに使えますが，パソコンなどがもつ自由な拡張性や，自由な組み合わせによる独自仕様のコンピュータにするには向きません．実用面や販売面からすると，パソコン向けのCPUアーキテクチャはx86仕様とPowerPC仕様しか存在しません．ほかには，特別に高性能なコンピュータの作成を目指したCPU以外は，すべて組み込み向けに開発されたCPUと考えてよいでしょう．

組み込み用のCPUも組み込むやりかたで大きく二つに分かれます．一つは独立したチップとして存在するCPUです．ゲーム機のPlayStationやPS2に使われたMIPS仕様のチップやカーナビによく使われているSH仕様のチップは，チップ単体で提供されています．

最近よく使われるARMなどはIPや仕様で提供されるCPUです．使うときは何らかのチップに組み込んで使います．図4のように組み込みに必要なすべての機能を一つのチップ上に集積するSoC（システム・オン・チップ）に向いています．CPUを組み込むチップはGA（ゲート・アレイ）などのマスク型のLSIでなくても，CPLDやFPGAに組み込むこともできるので，独立型のCPUチップと同じように，たった1台しか作らないときでも使うことができます．

● 組み込み用CPUはアーキテクチャの互換性が低い

組み込み用のCPUの最大の特徴として，アーキテクチャの上位互換性をあまり強く志向しなくてもよいという点があります．x86ではすでにあるOSやアプリケーションを捨てるわけにはいかないので，アーキテクチャ上の上位互換性は不可欠な要素です．これが，パソコン用CPUの開発にあたってかなり苦労するところです．

すなわちx86系のCPUは，すべてPC/ATのDOSレベルのプログラムで動く必要があります．もちろん互換性がないアーキテクチャを採用してもよいのですが，ほとんどのパソコン・メーカや一般ユーザが買わないと思われるので，Transmeta社のCrusoeなどx86アーキテクチャの一部を外部のプログラムで処

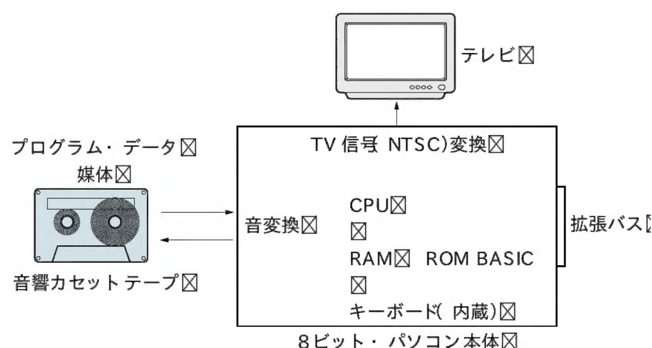


図2　8ビット・パソコンはOSなしで始まった

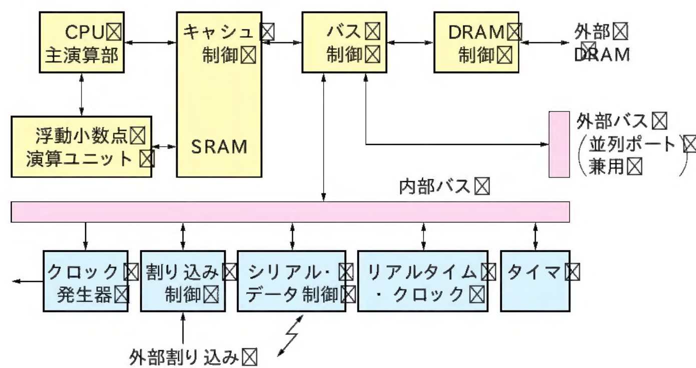


図3　代表的な組み込み用CPUチップの構成

表2　組み込みシステムに使われるCPUの一例

汎用CPU	組み込み向けCPU
x86，PowerPC，SPARC	8048，8051，78K0，V850，H8，SH，ARM，MIPS

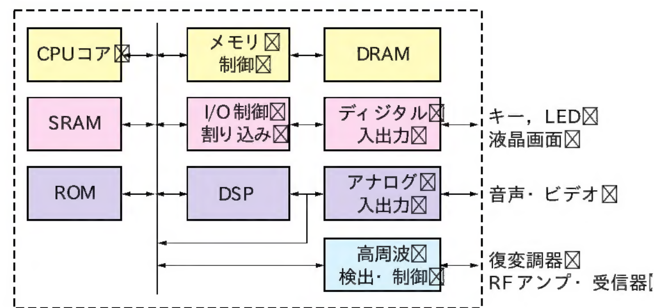


図4　システム・オン・チップの考えかた

表3　当初のPC/ATアーキテクチャ

CPU	80286
クロック	8MHz(当初6MHz)
主メモリ	256～512Kバイト
拡張バス	ISA
フロッピ・ディスク	1.2Mバイト
固定ディスク	20～30Mバイト(IDE)
外部インターフェース	非同期シリアル×2，プリンタ
キーボード	101(86)キー
グラフィック	EGA，640×350
マウス	シリアル非同期端子を使用
割り込み	15本
OS	(PC-)DOS

COLUMN

日本の8ビット・パソコン

Z80は1979年に日電が同社初のパソコンPC-8001に採用し，シャープからはROMなしのMZ80やアスキーが提唱したMSX型のパソコンなど，日本では8ビット・バス幅の16ビット・パソコンIBM-PCが1981年にIBM社から発売される前から大いに普及しました．

筆者はPC-8001の上で動くワープロを買い込んで，文書整理に使っていました．

理するチップ以外は，チップ自体が上位互換を保証しています．

一方，組み込み用のチップは，日立のSHなどはH8シリーズとの互換はもとより，SHシリーズ内でもその各チップが目的とする応用向けに応じてアーキテクチャに手が入れられており，同じハードウェアとプログラムが使えるようにはあまり考えられていません．

もう一つの特徴としては，組み込み用では，CPUチップは基板にじかにはんだ付けします．CPUの交換は不良品の発生時以外にはないからです．x86系のチップでもノート型のパソコンや一部の小型のデスクトップ型パソコンでは直付けになっていますが，デスクトップ型ではソケットを使って，不良交換時以外でも，性能向上のために交換できるのが一般的です．これはアーキテクチャ的にはCPUチップの信号のピン配置までも互換がなければいけないということです．

x86チップの互換チップのメーカでは，AMD社がK6-2まではこの完全互換路線を採ってきました．いまはIntel社自体がソケットに互換性がないx86チップを平行して供給しているので，チップの系列に応じたマザーボードが必要になりました．

4 PC/ATアーキテクチャと組み込みシステムの違い

よくも悪くも現在のパソコンのスタイルを決めたPC/ATアーキテクチャは組み込みシステムに向いているのか，その変遷と現状を組み込み用と対比してみましょう．

● PC/ATのアーキテクチャの誕生

IBM-PC(/XT)はそれなりに評価を受けましたが，Intel社から8086のアドレス限界である1Mバイトを16Mバイトに拡張した80286の提供を受け，それを使ったIBM-PC/ATが発表され，爆発的な拡がりを見せました．

PC/ATには表3に示されるように，パソコンとして使えるようなアーキテクチャが採用されました．これはIBM社が長年コンピュータ・メーカとして蓄えてきた力量が発揮されたと見てよいと思います．このアーキテクチャが公開されたこととIBM社のブランド力により，x86を使ったほかのパソコンを圧倒したものと思われます．

PC/ATは，今からみれば水準が低いアーキテクチャを採用していました．当時，Microsoft社から提供されたOSが8086の能力内でしか動かない，実アドレス中心のDOSであったため，大型機で培った技術はあまり利用されませんでした．DOSが動けばよいという範囲に限定されていて，かえってまとまりがよかったと考えられます．

日本語処理を行うためにハードウェアに漢字ROMを搭載するなど，日本独自のアーキテクチャとなったNECのPC-9801も，OSにはMS-DOSかCP/M-86が選択できるようになっていたため，IBM社のアーキテクチャと大差のないものになりました．

● PS/2でのMCAバスは瞬時の幻だった

PC/ATはアーキテクチャが公開されていたため，世界中でその互換機が作られました．IBM社の当初の目論見(もくろみ)のように，単独で世界制覇することはできず，むしろMicrosoft社が単独世界制覇を成しとげました．

PC/ATのアーキテクチャを実現するためのCPUチップの周辺回路は，図5のように寄せ集め的でした．74シリーズの標準論理IC，PLDおよびIntel社の8080A/8085用の周辺チップと非同期型のシリアルI/Oチップなどが使われていました．この機能はすぐにいくつかのチップにまとめられ，チップ・セットとしてPC/AT機やその互換機が作られました．

現在では多くのパソコンで，CPUに直結して主メモリとグラフィックの関係のインターフェースを受け持つノース・ブリッジとハード・ディスクやバスなどI/O関係のサウス・ブリッジの二つのチップにまとめられています．

その後，IBM社はアーキテクチャの開放で得るべき利益が散逸したと考え，新しいOSのOS/2が走るパソコンPS/2の発表とともに，ISAバスの後継として用意したMCAバスを開放せずに，バスに挿入する基板の製作者からもライセンス料を徴収しようと考えました．

この路線は，残念ながら互換機メーカの拒絶と多くのユーザの支持を得られなかったため，IBM社自身が売るパソコンも元のISAバスに戻る羽目になりました．PS/2での新しいアーキテ

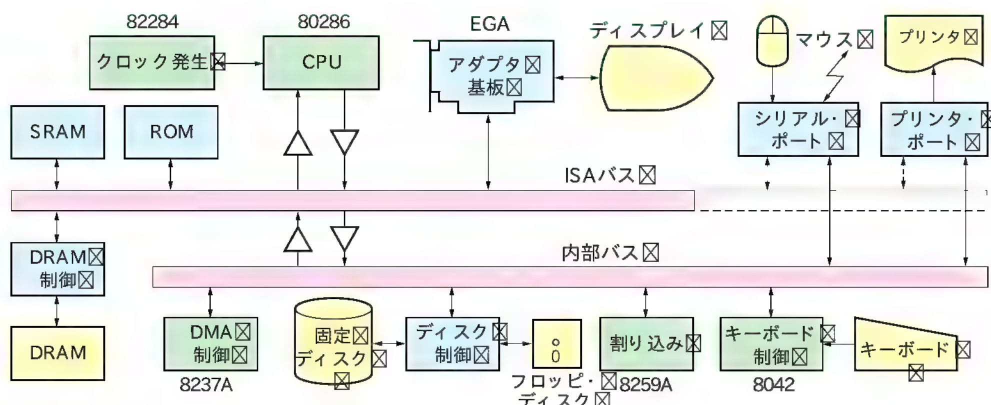


図5 当初のPC/ATアーキテクチャの内部構成

表4 主に割り当てられているPC/ATの割り込み

番号	内　容	番号	内　容
0	システム・タイマ	8	リアルタイム・クロック
1	キーボード	9	(PCIバスの割り込み)
2	IRQ8以下	10	(PCIバスの割り込み)
3	通信ポート2	11	(PCIバスの割り込み)
4	通信ポート1	12	PS/2マウス
5	(PCIバスの割り込み)	13	数値演算コプロセッサ
6	フロッピ・ディスク	14	プライマリIDE
7	プリンタ・ポート	15	セカンダリIDE

クチャはマウスとキーボードのインターフェースに残りました．

● PC/ATアーキテクチャの特徴

DOS上の制限と初期のアーキテクチャとの互換性を保つために，パソコンはIntel社のx86チップとともに大きな制約下で発展せざるを得なくなりました．

アーキテクチャ上の最大のポイントは，パソコンの拡張バスとしてISAバスを決めたことです．ISAバスは最近のパソコンには付いていませんが，グラフィック基板や種々のI/Oだけではなく，パソコン本体の基本機能である主メモリの拡張にも用いられました．

PC/ATアーキテクチャ上で意外と変わらないのは割り込みの制御です．割り込みの処理には8080A/8085用の周辺チップ8259Aが使われました．当初の設計は8259Aを二つ使い，一方の出力を他方に入れることで，表4のように15本の割り込み信号が使えるようになっています．この割り当ては今でも変わっていません．OSの側から見ると，これが変わってしまうと上位互換を維持するのが難しくなるようです．組み込みという観点からみるとこの割り込みの配置は固定的で扱いにくい状況です．また，PC/ATアーキテクチャのタイマ割り込みは1ms程度と間隔が長いのも特徴です．このため，リアルタイムの組み込みには向きません．

この割り込みはISAバスでは，割り込み信号が変化するエッジで検出していました．このため同じ割り込み番号に二つ以上の割り込み原因を割り当てることが難しく，結局一つのI/Oに一つの割り込みを割り当てるということになっていました．これにより，割り込みが必要となるような装置をパソコンにつなぐと，割り込みラインが不足するということが起きました．

また，優先割り込みを積極的に使っていないので，割り込み処理プログラムの中で暴走すると，キーボードやマウスが使えなくなり，ハードウェア的にリセットするしか復帰ができないというOSが作られる原因にもなりました．

● バスの高速化とシリアル化

ISAバスは8MHz程度のクロックで駆動されているため，信号転送の高速化が難しいという問題がありました．そのため，IBM社はPS/2仕様でMCAバスを策定したのですが，実際の高速化の動きは，ISAバスとコンパチブルなコネクタを使うEISAバスと80486の信号を生でISAバスの後に置いたコネクタに出すVLバスが提供されました．これらはおもに高速化を要求するディスプレイ基板用に使われました．

しかし，この考えはどちらもバス・コネクタのコンパチビリティを念頭に置いたものでしたが，やりかたに無理があったので，Intel社などが新たに策定したPCIバスに取って変わられました．ディスプレイ基板はすぐPCIバスに移行し，ほかのI/O基板も順次PCIバスに移行しました．PCIバスの割り込みはエッジからレベル・センス方式になりました．

ディスプレイ用の回路は，現在は図6のようにPCIバスから，ノース・ブリッジから出る高速なAGPバスに移されています．PCIバスになっても，パソコンに新しい外部機器を接続する際には，筐体を開けて拡張基板をネジ留めするという使いかたはそのままでした．これでは一般のユーザに向かないので，Windows 98の登場でOSに本格的なプラグ&プレイ機能を載せたのにともなって，Intel社などから発表されたUSBバスが使われるようになりました．

現在はUSBバスの高速化にともなって，多くのパソコン本体の外部に接続する機器が，USB経由で動くようになっており，その意味ではPC/ATアーキテクチャは当初の形からは大きく変わってしまっています．

現在，PC/ATアーキテクチャを組み込み用に使う場合，外

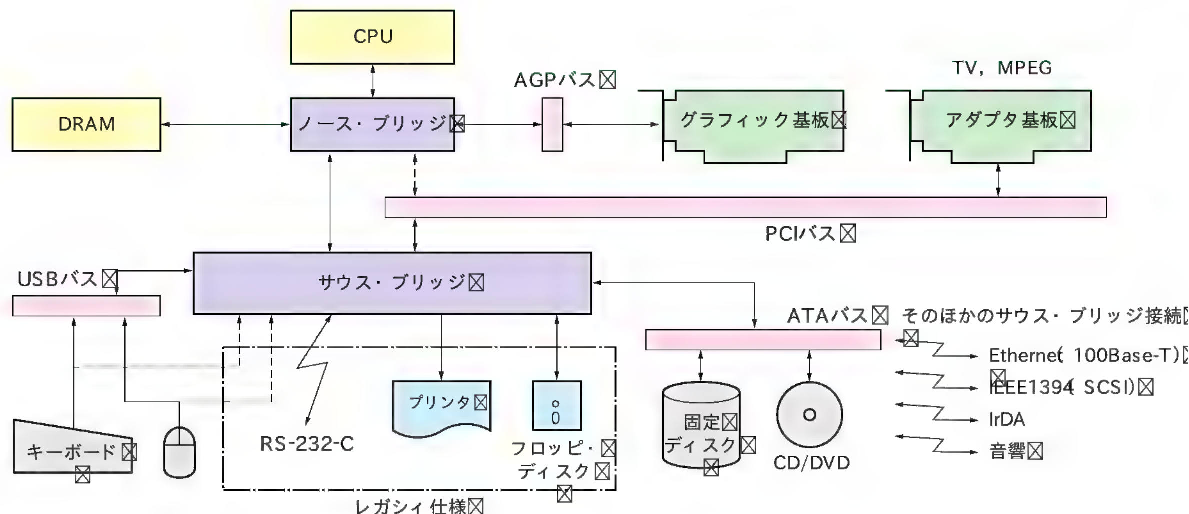


図6　現在の代表的なPC/AT互換機の内部構成

部との接続回路はPCIバスにアダプタ基板として実装するか，USB経由で接続することになります．ISAバスが使えた時代は，いろいろな便利な組み込み用のアダプタ基板が入手できましたが，今後はUSB接続の回路でそれらを実現していくようになるでしょう．それはPCIバスの基板を作成するより簡単に作ることができるからです．

割り込みも当然，PCIバス経由で取り込むことになります．すなわち，汎用の割り込み処理の中で処理する形になります．

● 組み込みシステムでのアーキテクチャ

組み込みシステムでは原理的には互換性を重視する必要がないので，原則としては機器ごとにハードウェアのアーキテクチャを決めることができます．

多くの組み込み向けのチップはCPU機能だけではなく，図3で示したようにコンピュータを構成する多くの機能をチップ内に取り込んでいて，シングルチップ・マイコンとして使えるようになっています．したがって，コンピュータとしてのアーキテクチャはチップ外部のメモリの量やアドレス配置などを決めるくらいしか，ユーザ側には自由度がありません．

汎用のCPUチップを組み込みに使う際には，アーキテクチャは本来比較的自由に決めることができますが，開発環境や保守を考えるとそれらが利用できる範囲に限った仕様しか採用できないことも明らかです．

現在では，4ビットや8ビットのチップでOSを使わない組み込みシステム以外では，何らかの組み込み用に加工したOSをもつのが一般的です．

組み込みシステムに要求されることが多いリアルタイム処理の機能を活かすための機能がCPUチップにあるとして，それをサポートするハードウェア機能は組み込みシステムではどうしても必要になります．CPUチップ上に搭載されているI/Oについては，これらの機能が付いているのが一般的なので，特別な外部機能や機器を付けるときに回路を作ることになります．

実際の組み込みシステムのOSでは，リアルタイム動作の保証だけでなく，データの保存やデータのやりとり，ディスプレイやキーボードなども使う場合があるので，それらを管理する機能をもたなくてはなりません．組み込みシステムのアーキテクチャはこのようなOSが動作できるように，割り込み機構などを完備する必要があります．

実際には，x86チップでもチップ自体には組み込みシステムで要求されるものと，同じ水準かそれ以上の機能が搭載されています．しかし，それらの機能はPC/ATアーキテクチャではある程度制限されているのです．

● 組み込みシステムでの外部機器の接続や遠方とのデータのやりとり

外部の組み込みシステムに機器を接続したり遠方とのデータのやりとりを行うとなると，どうしてもそのためのインターフェースが必要となります．そのような場合，ゲーム機や自動車の制御，カーナビあるいは家電機器のように生産量が多ければ，専用のアーキテクチャで接続することもできます．

しかし，生産量が少ない専用の制御機器の場合は，既成の外部機器や接続形式を使わざるをえません．なぜなら，独自路線では仕様を決めるところから始めて，開発環境や測定器さえも自分で用意しなければならないからです．

そこで，仕様が公開されている表5のようないろいろなインターフェースが使われてきました．この中には，当然ながらPC/ATの発達とともに提案された仕様が含まれており，どちらかというとそのような仕様を流用することで，組み込みシステムの開発にかかる労力や費用を下げ開発期間を短縮させることができます．また外部機器自体の価格やその開発および保守の費用も大きく削減できます．

LINは自動車用で，CANは産業・工業用，4～20mAは計

COLUMN

たくさん組み込めばよいというものではない

組み込みのチップにどこまでの機能を載せるかは難しい判断です．幕の内弁当も，吟味して並んでいるから美味しそうであるうえに食べやすいのです．

携帯電話にいろいろな機能を載せることが流行っていますが，壊れたり失くしたとき，あるいは現金決済機能が付いている携帯電話を盗られたときはどうすればよいのでしょうか?

電話番号帳を携帯だけが覚えていて，不便な思いをすることもあります．システムは，つねにトラブルが発生したときにお手上げにならない方法を用意しておかねばなりません．

表5　組み込みシステムで使えるいろいろな接続仕様

ディスプレイ	アナログRGB(DSub-15)，ディジタル(DVI)
パラレル	プリンタ・ポート(IEEE1284)，SCSI
ネットワーク	Ethernet
シリアル	シリアル・ポート(RS-232C)，USB，IEEE1394，IrDA
非パソコン用	電流(4～20mA)，RS-422，RS-485，CAN，LIN

表6　組み込み用にリアルタイムOSとして策定されたもの(一部)

OS仕様名	実用化例(連絡先)	規格元
μITRON	RX850(NEC)， TOPPERS/JSP Kernel(フリー)， REALOS(富士通)， μITRON(日立，東芝，三菱)， NORTi(ミスポ)， PrKERNEL(イーソル)， μMore(ACCESS)， ELX-ITRON(エルミックシステム)， ExRon(ファームウェアシステム)， μiPLUS(メンターグラフィックス)， TronForce(エーアイコーポレーション)， UDEOS(東芝情報システム)， I-right(パーソナルメディア)， ThreadX-μITRON(グレープシステム)	TRON協会
OSEK/VDX	osCAN(ベクター・ジャパン)， OSEK/VDX(日立)， PrOSEK(イーソル)， OSEKWorks(Wind River)， ERCOS(ETAS社)， Nu-cleus OSEK(メンターグラフィックス)	BMWなど9社
VxWORKS	VxWORKS(Wind River社)	自社
OS-9	OS-9(RadiSys社)	Microware社

装用の接続仕様です．これらは稼働環境面を考慮した方式が使われていて，これらのインターフェースにはパソコンなどではめったにお目にかかれませんが，特定の用途ではそれを使うことが標準となっていることもあり，組み込み用ではよく出くわす接続仕様です．

組み込み向けOSとWindowsの違い

● 百花繚乱のリアルタイムOS

パソコンだけを使っていて，この世界を初めてのぞく人にとって，世の中にWindowsとMacOS，Linux以外のOSが表6のように数多く存在していること自体が，希少生物でも見るような感じがすると思います．

日本の組み込み用のOSでよく使われているのはμITRONです．これはTRONプロジェクトの中ではかなりハードウェアに近い部分です．このOSはじつはカーネルの仕様だけですが，多くのメーカなどで実際のチップに対応した形に仕立て上げています．最大の特徴は仕様のすべてが公開されていて，ライセンス料の支払いがないことです．問題はトロン協会などが策定しているのは仕様だけで，特定の商品ごとに異なる形になるため，Linuxのようにそれらの間を越えて使い回すことができないという点です．

欧州の車メーカではITRONに対抗して，OSEK/VDXという名のリアルタイムOSの規格を決めて公開しています．この規格では，エンジンやサスペンション，エアバッグ，パワー・ステアリング，ABSなどの電子制御のためのデータ通信方法なども決めています．

ITORNもOSEK/VDXも仕様は公開されていますが，実用的に動くOSは商用のリアルタイムOSとして販売されています．

自社規格のWind River社のVxWORKSはいろいろなCPUチップへの供給を実現して，ITRONやOSEK/VDXの市場以外にもWindowsのリアルタイム化まで手を伸ばしています．

OS-9はMacOS 9とは違います．Macより歴史は古く，もともとMotorola社の8ビットCPUチップ6809用に開発されたROM化可能な組み込み用リアルタイムOSです．いまでは32ビット化されていますが，過去に6809が自動車などの制御に使われたこともあり，まだエレベータなど機械系の応用例には多く使われています．

● リアルタイムでないOSを組み込み用に使う

本来リアルタイム動作ができないOSでも，μsではなくms程度の時間的なゆとりがある応用分野なら，各プログラムに提供されているスレッドやジョブの切り替えが間に合う範囲で，組み込み用に使うことができます．

LinuxやFreeBSD，NetBSDなどをそのまま高速の組み込み用に使うという選択をする際は，ある程度のリアルタイム処理の実現のためには，かなりのくふうが必要となります．実際に何もくふうしないままのFreeBSDで，正弦波の一周期を256分割してプリンタ・ポートから出したところ，数Hz程度の周波数までしか出せませんでした．大体1ms程度の処理速度です．

ただ，これらのUNIXライクなOSは内部の構造が公開されているため，図7のようにμITRONやVxWORKSの上に載せるという形で高速のリアルタイム性を実現することも可能です．

ところで，これらのUNIXライクなOSでは，ライセンスの

問題をよく管理しておかないといけません．GPL のプログラム・コードを使うときは，完成した商品のプログラムの構成をうまく分けておかないと，商品のノウハウが盛り込まれたプログラムまで公開する羽目になることがあります．BSDライセンスはこのへんが比較的自由なので使いやすいともいえます．

Microsoft 社も Windows CE という組み込み用というか，PDA 用を目指した OS を出しています．これはスタンド・アローン型の Windows との共存を売りにしてはいますが，ライセンスや内部仕様の隠匿性などについては何も解決されていません．

● リアルタイム性はカーネルの作りかたで決まる

組み込み用の OS と Windows との違いの詳細は後で説明することにして，なぜ Windows などの OS が組み込み用には向かないのかということを考えてみます．

組み込み向けあるいはパソコンや大型機を問わず，OS には最低限必要な機能と機器を使いやすくするための機能があります．前者のような OS の核となる部分をカーネルといいます．

OS はカーネルだけでは手足がないので，最低限コンピュータとして外部とのデータのやりとりを行う機能を必要とします．この部分はハードウェアのアーキテクチャによって変わるので，いろいろなハードウェアで共用できる汎用の機能と，特定のアーキテクチャに特化した機能に別れます．この汎用機能も含めてカーネルと考える人もいます．

一方，後者の機能は使う側で欲しいものは何でも含まれるので，デスクトップ用の Windows などはどんどん肥大化していきました．本来ならアプリケーションであった Internet Explorer を内部のファイルを見る機能と意図的に関連づけてしまっているのもこのような考えです．

このような選択肢がある中で，組み込み用としてリアルタイム動作を保証しようとすると，カーネルをそれに則した形で作らなければなりません．組み込み用のシステムでは外部からのいろいろな種類の頻繁なサービス要求に即時に対応することが基本となるのです．

重要性に順序を付けたサービス要求とそれを処理するプログラムであるタスクの切り替え(タスク・スイッチ)が高速なことが，カーネルの機能の中で最重要機能と見なされることになります．

これに対してファイルの処理や対人処理が主であるパソコン用の OS では，多重処理やバック・グラウンドでの高速大量データ処理においては高性能を発揮しますが，頻繁にあちこちにデータを振り回すようなリアルタイム処理に対応するようにカーネルが作られていません．

● 組み込み用 OS が必要なわけ

自動車などは図 8 のようにエンジンやタイヤの 1 回転ごとに，最低 1 回は制御システムに対して何らかのサービス要求があると考えて置かねばなりません．

CPU のクロックが遅かった時代は，このような高速なサービス要求に対しては，単純な割り込み処理プログラムで高速応答してきました．すなわち一つのおもな処理要求に一つの CPU

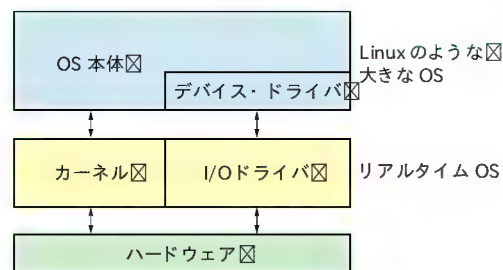


図 7　リアルタイム OS 上に普通の OS を乗せる

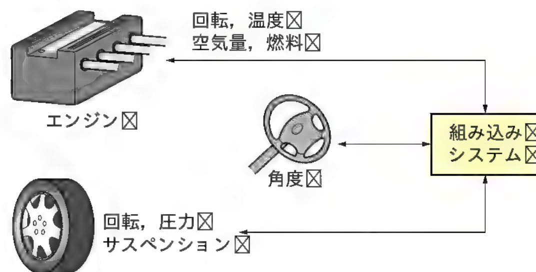


図 8　高頻度にサービス要求が来る組み込みシステム

チップを割り当ててやれば，エンジンなどの高速な処理でも間に合うわけです．

筆者も 10MHz の Z80 で 38.4kbps のメモリ付き双方向通信バッファを組み込み用に作ったことがありますが，十分すぎるほどゆとりがありました．このアプリケーションではタスクの数が限られているので，割り込み処理プログラムでタスク・スイッチを行うことなく，各サービス要求ごとに割り込み処理ルーチンで直接処理したからです．

しかし，単純な割り込み処理では，次のようにいろいろな問題が発生しました．

1) 一つのタスクすなわち，簡単な家電製品などその部分に限った制御やデータ処理しかできない
2) 自動車などでは全体を一つのシステムとして車内の相互の

RTOS は処理効率はよくない

RTOS(リアルタイム OS)は頻繁にタスクの切り替えが行われるために，実際の CPU 時間のかなりの部分をこの切り替え処理に取られてしまうことがあります．

限られた CPU 時間でできるだけたくさんのデータを通したり，処理するためには，旧態依然ですが割り込み処理プログラムですべての処理を行って，それをアセンブラで書くのがもっとも高速になります．

でもそんな職人芸に頼るよりも，クロックを上げて RTOS を使うほうが，開発も検証も楽です．

COLUMNS

バーチャル体験では技術力は上がらない

パソコンの発達で，多くの事がらがシミュレーションできるようになりました．大型のハイビジョン・テレビを見ていれば世界中旅行に行った気分になれるわけですが，どうもいまひとつ感覚が違います．

運転シミュレータでも，ある程度の運転技術は身につきますが，路面の感覚や前の車の運転者が携帯で電話したりカーナビでテレビを見ながら運転しているのを感知して，事故を避けるために車間を空けるというような技術は，実地で学ぶことにはかないません．

開発作業も実地にたくさんこなさないとうまくならないようです．

関係を制御・管理する必要がある．すなわち，いろいろなサービスを同時かつ全体的にこなす必要がある

3) 単純な割り込み処理だけでは，各応用例ごとにプログラムの開発を行わなければならない．せっかく苦労して作成した割り込み処理プログラムの使いまわしができない
4) 取り扱うデータの量が増え，4ビットや8ビットのCPUでは限界が出てきて，16ビットや32ビットのCPUが主流となった．このためアセンブラで割り込み処理プログラムを作成することが困難になり，Cで開発を進めるのが普通になった
5) EthernetやUSBのような汎用のインターフェースで外部と接続することが普通になったので，標準提供されるドライバなどが必要となった
6) 開発をターゲット機器の上で行うのではなく，ほぼ最終的なデバッグまでクロス開発環境で行う，平行同時開発があたり前になった

など，いろいろな要因が重なり，組み込み用のCPUチップへのOSの導入が不可欠になりました．

組み込みシステム開発の流れ

組み込みシステム・ビジネスは時間との勝負です．その中でいかに安価に安定に仕上げるにはどうするかを考えてみます．

● 組み込みシステムは人を食う

組み込みシステムの開発は，ハードウェアと，その上で動くソフトウェアの開発の同時進行を普通とします．もし時間があれば，

1) ハードウェアのアーキテクチャを自分が気に入るように作る
2) ハードウェアの上に最適化したリアルタイムのOSを載せ，外部の機器とのインターフェースを完全にチェック
3) アプリケーションを順番に作っていく

というステップを踏んでいくこともできます．

実際には，8ビットの時代まではこのようなやりかたを採用していても十分間に合いましたが，いまでは学生が学部4年の卒論から始めて，院の卒業までかけて楽しむときくらいでしか採用できません．

開発に要する仕事量と組み込みシステムを試験する手間が増えただけでなく，能力的にも一人でハードウェアからOS，実際の制御プログラムまでを手がけることができなくなりました．

過去にはいたそのような人材は，今は経験するべきことや学習内容が多くなりすぎたために，育成することが不可能になってしまいました．すなわち，一つの組み込みプロジェクトをたくさんの人で同時に進行させなくてはならなくなったのです．

● 組み込みシステムの開発は開発者のインターフェースがすべて

一人で組み込みシステムを開発できない以上，共同作業が必要となります．開発の人員が全員同じ部屋にいる状態ならよいのですが，部屋が違ったり建物や場所が異なると，開発作業はたいへんな手間になります．簡単なことでも開発者どうしのインターフェースが取れていないままで開発を進めてしまうと，あとでお互いの間が接続できなくなり，どちらかが大きく変更することを余儀なくされます．

このようなことを避けるためには，毎日定期的に電子メールなどでやりとりするという初歩的な方法だけではだめで，開発意思の同期を強制的に取るCASEツールのようなやりかたを，ハードウェアからすべてに適用する必要があります．

開発作業は，対象の大きさにもよりますが，次のような労力配分となると考えておいたほうがよいようです．

1) 仕様の作成と擦り合わせ　30%
2) プログラムや回路の設計　10%
3) 開発環境の整備・習熟　10%
4) 動作検証・デバッグ作業　50%

この中で，1)と4)の段階で開発者どうしの密な連絡とやりかけの仕事の進行管理が必要となるわけです．

● 具体的な開発スケジュールの例

開発は同時進行を旨とするので，図9のような簡単なパート図程度の開発のチャートを作成してから，開発の方針を決定したほうが便利です．

ここで時間的なネックになるのはICの開発です．最初の開発会議では，コストと時間のバランスを見てICをFPGA/CPLDにするか，ゲート・アレイにするかを決定します．

全体としては外部仕様の確定時期が，最後の試験工程に大きく響いてきます．これは，この例だけでなく多くの組み込みシステムで起きる現象です．

とくに特定の機能をICで実現するかプログラムで実現するかという選択は，システムの構成に大きく影響するので，仕様作成の段階で決定しておいたほうが，あとでよけいな労力を使わずに済みます．筆者の経験では，なるべく高速なCPUと簡単なハードウェアを使って，プログラムで処理するようにしたほうがうまくいくことが多いようです．

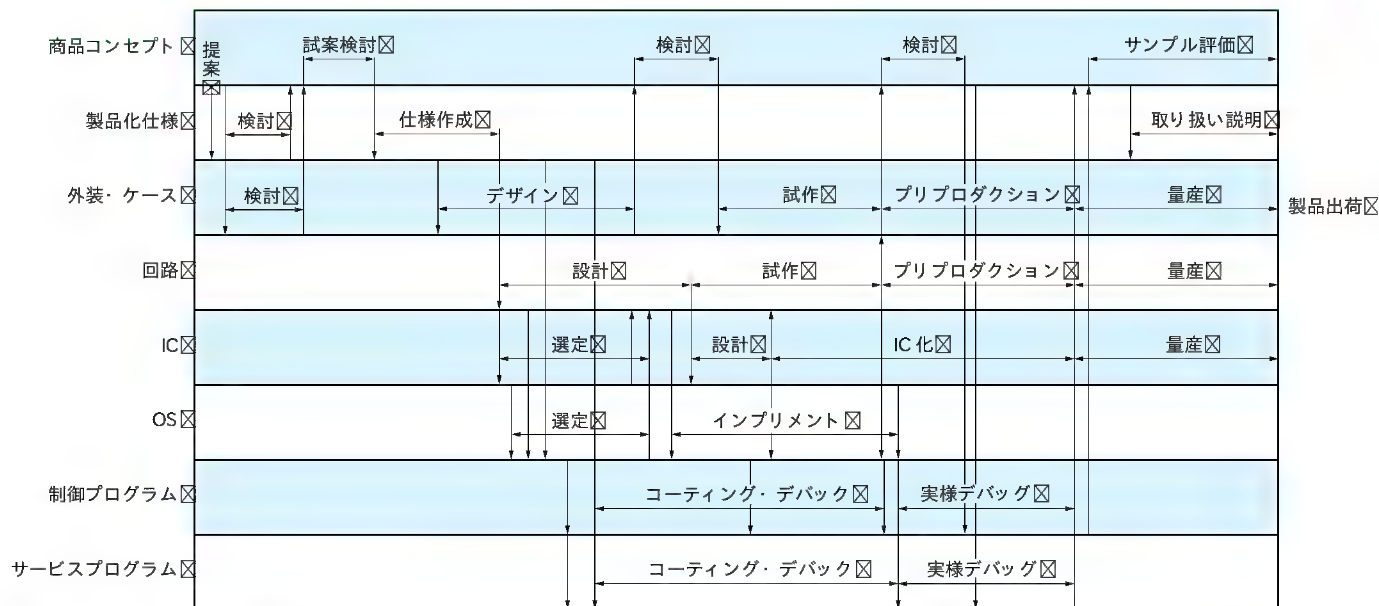


図9　開発工程のパート図の例

組み込み特有の価値観について

組み込みの世界での価値観は，それが使われる箇所での価値観に左右され，いってしまえば，「速ければ正義」とは限らない部分があります．

● 何をもってCPUを評価するか

組み込みシステムについて，いろいろと述べてきましたが，いちばん重要な点について考えてみます．それは，システムの側で組み込みCPUに何を求めているかということです．

CPUの評価はいろいろな要因で決まりますが，お互いに相反する要求もあり，それらの間のトレード・オフを考えねばならないこともあります．評価の基準を以下のようにざっと挙げてみると，いくつかのグループに別れます．

1) 速く動くこと
2) 機能がたくさんあること
3) 既成のOSやアプリケーションが豊富なこと
4) ソフトウェアの開発が容易なこと
5) 消費電力が小さいこと
6) 寸法が小さいこと
7) 小型であること
8) チップ単体が安価であること
9) 壊れにくいこと
10) 実装や取り扱いが容易なこと
11) 安定供給されること
12) 特別の契約（特許料などの支払いや対外販売への制約）が不要なこと
13) 企業なら会社の方針に合うことや幹部の了解が得られること
14) 開発担当者の意欲をそそったり趣味に合うこと

● パソコン用のCPUは速さだ

パソコンでは1)～4)のあたりが重視される傾向にありますが，組み込みシステムとなると，それだけでCPUチップを選ぶことはできません．量産品として製品に組み込む際は，9)～14)が比較的重要なポイントになります．たとえば，H社の技術者は13)によってSHシリーズをまず最初の選択肢に考えねばならないことになるわけです．

5)～11)は，原理的にはどのCPUチップに対しても出る要求ですが，その中でもお互いに優先度を考えねばなりません．基本的には手に持って運ぶものについては，この要求は強く出ます．

● 量産品への組み込みCPUは強いていえば価格

たとえば多機能を目指す携帯電話などは，どうしても技術的に最先端のCPUチップが必要になります．ただ，生産量がパソコンなどより多いので，ある程度は価格要求を満たすことが可能なチップが入手できます．

組み込みシステムの価値観は，作られた機器の生産量とバージョン・アップの頻度でも変わります．携帯電話はいくら買い替えが早いといっても一年の長さです．しかし，その（電柱などに付いている）基地局はサービスが変わると月単位でプログラムを変えるだけではなく，ハードウェアにも手を付けなくてはならないこともあります．その場合は，そのようなことが可能なIPとしてのCPUと書き替え可能なFPGAとの組み合わせが有力になります．

先日H社が商品タグ用のチップを10円台で供給すると発表して，たいへん話題になりました．ユビキタス時代の到来をもたらすこのようなチップは，性能は当然ながら現時点では価格が採用決定の最大基準です．

COLUMN

一品料理から崩れてきた日本の製造技術

いま日本で最大の問題となっているのは，一品料理をこなせる技術者が，だんだんと高齢化したうえに，工場の海外進出などで仕事も減ってしまったので，技術者を育成する環境がなくなってしまったことです．

技術の継承だけでなく，新しい技術を育てることも難しくなっています．日本の製造技術の中で，組み込みシステムの制御などに関しては，一品料理が引っ張ってきたという経緯があります．本誌の読者にも大いに期待したいところです．

● OSと開発環境は開発担当者の趣味に任せるのが一番

開発環境は，使用するCPUチップとOSが決まれば，ほぼ自動的に決まってしまいます．Linuxなどではクロス開発環境が十分に利用できるので，開発者の裁量で言語さえも選ぶことが可能です．

実際は開発を管理・依頼する側のあまり根拠がない理由で，OSなどは決まってしまうことが多々あります．とくに日本の企業では，自社の特徴をどうやって出すかということに自信がなくて大勢に流されやすい幹部が多いので，業界みんなが使っているという理由で，PDAにWindows CEやPalmOSを導入してしまうところが多いようです．

若手の技術者の側でも，責任を取るのがめんどうなので，上の人が無難な案を出すとそれに安易に乗ってしまう傾向にあります．このような商品の売れ行きがパッとせず，利益に貢献できなかった場合でも，決定の過程がうやむやなので責任も不在となってしまいがちです．

本当は若手の技術者に自由にものをいわせて，納得がいく結論が出たら，幹部が責任を取る代わりに，担当者は本気でOSや開発環境にほれ込んでもらうのがいちばん効率がよく，かつよい製品ができます．

表7　製造・営業用機器の法定耐用年数

機器名	年数
自動車製造設備	10
エンジン，同部分品または車両用電装品製造設備	10
自動車車体製造または架装設備	11
車両用ブレーキ製造設備	11
そのほかの車両部分品または附属品製造設備	12
プレス，打抜き，しぼり出し，そのほかの金属加工品製造業用設備	12
電気計測器，電気通信用・電子応用機器，部分品製造設備	10
産業用，民生用電気機器製造設備	11
電気機器部分品製造設備	12
金属加工機械製造設備	10
機械工具，金型または治具製造業用	10
建設機械，鉱山機械または原動機付車両製造設備	11
金属製洋食器またはかみそり刃製造設備	11
農業用機械製造設備	12
鉄道車両または同部分品製造設備	12
鋼船製造または修理設備	12
自転車または同部分品もしくは付属品製造設備	12
そのほかの産業用機器または部分品もしくは附属品製造設備	13
事務用機器製造設備	11
金属製家具もしくは建具または建築金物製造設備	13
食品用，ちゅう房用，家庭用，サービス用機器製造設備	13
歯車，油圧機器そのほかの動力伝達装置製造業用設備	10
ねじ製造業用設備	10
鋼製構造物製造設備	13
前掲以外の機械器具，部分品または付属品製造設備	14
機械産業以外に属する修理工場，工作工場用機械設備	14
そのほかの金属製品製造設備	15
パーソナル・コンピュータ(サーバ用を除く)	4
そのほかの電子計算機	5
プリンタ，コピー機，ファクシミリ，そのほかの事務機器，ディジタル・カメラ，ビデオ・カメラ	5
ソフトウェア(販売用の原本を除く)	5

● 一品料理は作りやすさと長持ちがポイント

組み込み用途といっても，せいぜい数台から数百台程度の組み込みシステムは，そのアプリケーションの種類の多さから見ても，いちばん技術者の数を要求する分野です．たとえば，昨年末から始まった地上波ディジタル放送用の機材などは，量産品という水準の数は不必要ですが，パソコンと同じような要求事項の上に，12)を除く9)～14)の条件が満たされる必要があります．

このように生産量が少ない組み込みシステムは，ほとんど注文生産に近いので，一品料理といわれています．どちらかというと，価格より性能や納期・作りやすさが重要視されます．さらに，恐ろしいことですが，10年程度ではまだ現役の組み込みシステムは常識です．

先日も，とあるシステム管理機器に組み込まれた基板を保守できるかどうか検討した際，CPUの製造年月には1987年の刻印がされていました．工場の機械などの法定耐用年数は短くても10年，長ければ15年です．パソコンやソフトウェアでさえも，税法上は4～5年の耐用年数になっています(表7)．動かなくなったといっても，どんどん買い替えるわけにはいかないのです．顧客側では，税法上の償却が終ってから利益が出るという感覚ですから，15年や20年でそれを制御するシステム側が保守不能になってはいけないのです．

おわりに

組み込みの世界は，いってしまえば山脈のようなものです．スタンド・アローンなパソコンの世界は，だれでも一度は登る富士山のような独立峰と考えるとよくわかります．独立峰はよくも悪くも目立つうえに数が限られています．また，その山頂に登るのは簡単ですが，上に立つと偉くなったようでとても爽快です．

組み込みの世界は山脈ですから，次から次へと山頂が続き，とくに目立つということはありません．隣の山頂へは簡単に移動できますが，道を選びまちがえると遭難してしまうこともあります．この世界へ首を突っ込むには，冬山を縦走するベテラン登山家と同じような準備万端の心構えが必要です．

いかい・くにお　(株)エム・アイ・ベンチャー

2. 地の底から宇宙の果てまでを支える

社会のインフラ——組み込みOS

藤倉 俊幸

以前は小さなCPUと数Kバイトのメモリで動いていた組み込みシステムだが，ネットワークやリアルタイム処理などへの対応の必要性から，OSが搭載されるようになってきた．近年の要求仕様の増大はとどまることを知らず，この流れは続くであろう．

組み込み機器にOSを搭載するとき，そこには普段使われているデスクトップ向けOS——Windowsを使わずに，“組み込みOS”と呼ばれる非Windows OSが採用されることがある．もちろん，組み込みでWindowsを採用する動きもある．Windowsであれば，多くの技術者が存在し，既存のクラス・ライブラリやノウハウなど，有形無形の資産が活用できるというメリットがある．

それにも関わらず，あえて組み込みOSを採用するのはなぜなのか．もちろん，そこにはそれなりの理由が存在する．（編集部）

はじめに

まず，組み込みOS（Operating System）の特徴について考えてみる．「OS」とは，基本ソフトウェアなどと呼ばれることもあるように，ソフトウェアである．どのようなソフトウェアかというと，ほかのアプリケーション・ソフトウェアを走らせるためのソフトウェアである．ことばとしてのOSの元祖は，IBM704のSOS（Share Operating System）だといわれている（写真1）．その後，モニタ・システムやエグゼクティブ・システムなどと呼ばれていたものが統合され，今のOSという概念ができあがった．

今でもタスク・スイッチングを中心とした機能のみを提供する場合には，OSと呼ばずに「モニタ・プログラム」と呼んだりする．たとえば，「ITRONはモニタ機能のみでI/Oはサポートしていない」のように使われる．



写真1　元祖OSのIBM704（1954～1960）

さらに，OSというと，普通はアプリケーション・プログラムを走らせるだけでなく，入出力を抽象化するサービスを提供したり，ユーザ認証を行う機能もある．しかし組み込みOSでは，ユーザ認証をサポートするのはまれである．また，入出力もデバイスが多様すぎて汎用OSのように抽象化しきれないことが多い．つまり，

組み込みOS＝モニタ[注1]＋デバイス・ドライバ
＋ミドルウェア[注2]

が基本的な構成である．ここでいうデバイス・ドライバとは，WindowsのWDM（Windows Driver Model）に対応するようなOSが提供するI/Oモデルのことである．先の「ITRONはモニタ機能のみでI/Oはサポートしていない」の意味は，（初期の）ITRONではデバイス・ドライバについては何も規定していなかったという意味である[注3]．I/Oをモデル化してドライバの構造を定義することの利点は，デバイスの制御を統一的に行えることである．さらにデバイス・ドライバをコンポーネント（独立した部品）として流通させることもできる．

このようなITRONに対して，UNIXの流れをくむ組み込みOSはドライバ構造を持っていて，一般的なデバイスであれば接続することが容易である．Linuxが注目される理由の一つはドライバの豊富さである．ドライバのほかに，オプションでファイル・システムやネットワークなどをミドルウェアとして追加できる．

そして，モニタは，

モニタ＝カーネル＋システム・コール

と分解して考えると理解しやすい．メモリ保護機能，デバッグのサポート，スケジューラなどの各種付加機能をコンポーネントとしてカーネルに追加・変更できるものもある．最近の組み

注1：モニタというと，セマフォとデータをいっしょにして排他制御する機構をさすこともある．ここでは，そのモニタではなく，モニタ・プログラムの意味である．
注2：ミドルウェアとは，OSとアプリケーションの間に存在する特定の機能を提供するソフトウェアである．
注3：μITRON 4.0ではデバイス・ドライバ設計ガイドラインによってデバイス・ドライバの作成法が規定された．しかし，具体的なドライバ構造が規定されたわけではない．http://tron.um.u-tokyo.ac.jp/TRON/ITRON/GUIDE/device-j.html

込み機器は多様化してきたが，さすがに仮想記憶をサポートするものは少ない．

● そもそも「組み込み」とは？

「組み込み（Embedded）」ということばは，コンピュータの用途を表している．たとえば，表1に示した機器の中に組み込まれて使われる．ただし，組み込み業界の実態がはっきりしない．また組み込みシステムとは何かがはっきりしないので，組み込みOSも同様ではっきりしない．それでも，なんとなく暗黙の定義があり，業界関係者の間では話が通じている．

組み込みソフトウェアのことを，「ファームウェア（firmware）」と呼ぶこともある．ソフトとハードの中間の硬さという意味でfirmを使う．組み込みソフトウェアのうちROMに焼かれたソフトウェアのみをファームウェアという場合もある．PCの中にもファームウェアはあって，普通はBIOS（Basic Input/Output System）と呼ばれている．PCに電源を入れてWindowsが立ち上がるまではファームウェアが走っている．だから，組み込みエンジニアがいないとPCは作れない．また，WindowsのWDMに従ってデバイス・ドライバを作る際には，組み込みソフトウェアのスキルが必要になる．組み込みエンジニアの仕事は，PCでいえばBIOSのレベルからアプリケーションを作ることだと思えば良い．

● 日本における組み込みの立場

本年の2月上旬に行われた形式仕様に関するシンポジウム(1)で，経済産業省の職員が3月に5,000社程度にアンケート調査を行って，組み込み業界の実態を把握したいといっていた．つまり，組み込み業界は，今までは国でさえ統計資料を持っていなかった雑多な世界なのである．この記事が出版されるころには，何か結果が出ているかもしれない．IPAのホームページ(2)をのぞいて見る価値はあると思う．

これまでの日本ではハードウェアが主役で，組み込みソフトウェアは，OSも含めてハードウェアの陰に隠れていたように感じられる．そのため，ゲーム業界や家電業界などのハードウェアによる分類に基づく統計はあったが，そこで横断的に利用される組み込みソフトウェアに関するまとまった情報はなかったのである．1年間にどの程度の開発予算が組み込みソフトウェアに使われているのかさえ，だれも知らない．

しかし，ここに来て組み込みソフトウェアの重要性が認識されつつある．製造業の国である日本を支えているのは，組み込みソフトウェアの開発力だという人もおり，日本は組み込みソフトウェアの開発力があるとする立場を取っている．これらの人たちは，開発力を維持するために何か施策が必要だという．しかし一方で，欧米からかなり遅れているとされる中国や韓国にさえ，すでに遅れを取っているという人もいる．これらの人たちは，開発力を高めるために何か施策が必要だという．実際のところ，現状の開発力は不明であるが，いずれにしても組み込みソフトウェアは重要で，何かしなければならないという認識に変わりはない．それで日本でもカーネギーメロン大学にあるようなソフトウェア工学センタを作って，そこには組み込みソフトウェアを扱う部門も設置されるようである．

現状がわかりにくい組み込みの世界を，4月からデビューする人たちに解説することが今回の特集の意図である．「Interface」と書いてあるこの雑誌を手にしたあなたは，この世界の入り口に立っていた．そして，この章まで読んだことで，すでに第一歩を踏み込んでいる．統計資料がないということは大人の怠慢ということもあるが，かなり多岐に及んでいて（表1）かつ動的に変化している（図1）ことも理由として挙げられる．この不思議の国はかなり広大で絶え間なく変化している．そして，チャレンジングでやりがいのある世界だ．優秀な人材を求めている世界でもある．

表1　組み込みソフトウェアの例

分　類	製品例
白物家電	炊飯器，電子レンジ，冷蔵庫，洗濯機，乾燥機，エアコン
情報家電・AV機器	テレビ，ビデオ，ディジタル・カメラ，オーディオ機器
娯楽/教育機器	ゲーム機，電子楽器，カラオケ，パチンコ
個人用情報機器	PDA，電子手帳，カーナビ
パソコン周辺機器	プリンタ，スキャナ，ディスク，CD-ROMドライブ
通信端末	電話機，留守番電話機，携帯電話機
ネットワーク設備	交換機，PBX，ネットワーク・ルータ，ハブ
業務用機器	業務用データ端末，POS端末，自動販売機
運輸機器	自動車，信号機，鉄道車両/制御，航空機，船舶
工業制御/FA機器	プラント制御，工作機械，工業用ロボット
設備機器	ビル用照明，ビル用空調，ビル用電力システム，エレベータ
医用機器/福祉機器	血圧計，心電計，レントゲン，CTスキャナ
宇宙	人工衛星，ロケット

オプションだらけの世界

この世界では，CPUがIntelで，OSがWindowsというよう

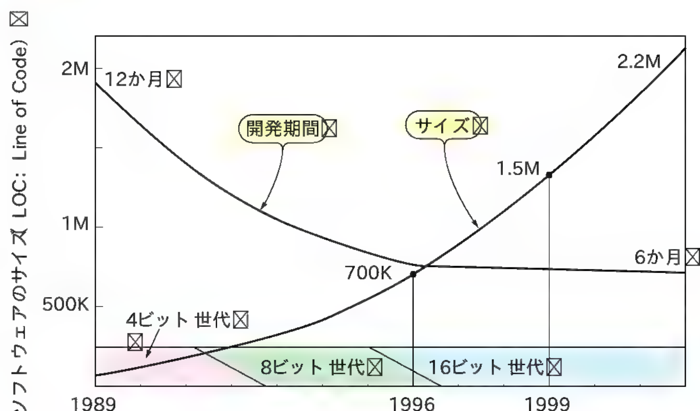


図1　携帯電話の場合のソフトウェア・サイズ変化

な定番となる環境は存在しない．外国に出張して朝食を注文するときのように（写真2），ハムにするかソーセージにするかベーコンにするか，卵はゆでるのか焼くのかポーチド・エッグにするのか，焼くのなら焼きかたはどうするのか，いちいち聞かれるのと同じでオプションづくめになっている．しかもなじみ深い肝心の卵焼きが選択肢にない．つまり，学校ではほとんど組み込みソフトウェアについて教えていない．CPUはダブルでお願いします．一つはITRONを載せて，もう一つはLinuxでお願いしますという感じである．OSなしという選択ももちろん存在する．

どうやったら，外国のホテルで粋（いき）に朝食を注文できるのか，ということがビギナの当面の目標である．必要なのは知識と経験と度胸だろう．知識と経験は追々身に付けるとして，とりあえず度胸のない人はちょっとやせがまんして，開き直って前へ進むべきだ．「厚焼き卵を食わせろ，何だ知らないのか」といってみるのも良いかもしれない．オプションだらけの世界では，人をまねるよりも独自の自由な発想のほうに価値があるのだから．

● ポストPC OS

最近，組み込みの世界に入ってきた人と話をしているとき，組み込みOSを「PC以外で使われるOS」と定義して話をしていたが，どうも話が合わない．再度，話のすり合わせを行うと，その人はPC以外の情報機器のOS，たとえばPDA[注4]や電子手帳のOS（ポストPC OS[注5]）の話をしていた．表1の個人用情報機器と娯楽機器に限定した話と，表1全体を対象にした話ではとんちんかんなやりとりになるのは避けられない．このようなことは良くある．つまり，ベテランでもときどき軌道修正しながら話すしかない．しかし，話し始めないことには先に進めない．

ポストPC OSといった場合，個人用情報機器と娯楽機器だけになる場合が多い．WinCEやLinux，PalmOS[4]，SymbianOS[5]などが該当する．これらは組み込みOSのほんの一部である．たとえば，携帯電話の場合，電話帳とかカレンダなどの機能はSymbianOSかLinuxを使用しているが，通信機能については別のCPUを用意してITRONを載せている場合がある．つまり，ポストPC OSと組み込みOSとの組み合わせということになる．

いつまでこのようなアーキテクチャの携帯電話が存在し続けるかはわからないが，組み込み全体の開発件数では情報機器以外が格段に多いことは変わらないだろう．だが，金額ベースでは情報機器の比率はますます増えていくと思われる．現在注目されているユビキタス・コンピューティングは，組み込みのようで組み込みでない新しい世界なのかもしれない．その世界に対応するのがポストPC OSかもしれない．

● 組み込みOSの定義

そんなわけで，何でもありの組み込みOSとは何かをむりやり定義すれば，CPU（ハードウェア）とアプリケーション・プログラムをくっつける「糊（のり）」のようなものということになるだろう．でもこれだけでは，「組み込み」を取り立てて付ける必要はない．ただOSといわずに「組み込み」を付ける理由は何だろうか．そもそもどうしてオプションだらけになっているのだろうか．

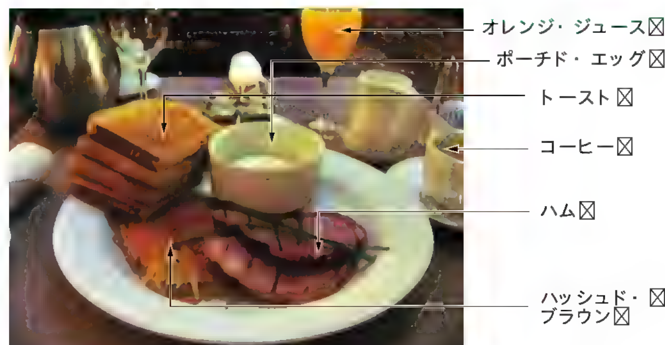


写真2　ある日の朝食オプション

組み込みOSの特徴

組み込みシステムは多岐にわたっていて，生きている世界がPCとはまったく違う．Windowsは机の上，組み込みOSは地の底から宇宙の果てまでを支える社会のインフラとして，縁の下の力持ちとして生きている．生物も生きている世界によって基本的な構造が変わってくる．組み込みOSも同様でいろいろなタイプがあって，その結果，オプションづくしになっている．

たとえば，春先の水を張った田んぼにいつのまにか現れるホウネンエビ（写真3）は，よく見ると，足が11対あってそれを使って泳いでいる．体長は2cmぐらい．また，もう少し小さいミジンコ（図2）は1mm程度で，これも足で泳ぐ．ところが，さらに小さくなると足は使えなくなって繊毛を使って泳ぐようになる．たとえば，0.2mm程度のゾウリムシ（図3）は繊毛を使う．そして，さらに小さくなるとまた別のメカニズムで泳ぐ（図4）[注6]．生きている世界により，水の流れに影響を受けるのか，水の粘度に影響を受けるのかで，泳ぐという動作に対する主要な要因が変わってきて，足を使うか，繊毛を使うか，鞭打つのか，回転させるのかといった最適なデザインが変わってくる．だから，組み込みOSを機能やメカニズムで定義してしまうと，本来多様であるべき組み込みシステムに対して不必要な制限を課してしまうのではないだろうか．老子曰く，「道の道とすべきは，常の道にあらず」に近い心境である．したがって以下は単なる現状の分析にすぎない．

● ハードウェア制約と時間制約

OSも生きている世界によって構造が変わってくる．Windows

注4：PDA（Personal Digital Assistants）：ピーディーエー．携帯性を重視した情報端末のこと．
注5：PC以外の携帯電話などの情報機器を使用する人が最近は増えてきた．これらのPCの後に続く機器に使われるOSを，ポストPC OSと呼ぶ．
注6：バクテリアの泳ぎかた：http://info.bio.cmu.edu/Courses/03441/TermPapers/2001TermPapers/chemotaxis/caroline.html より．



写真3　ホウネンエビ

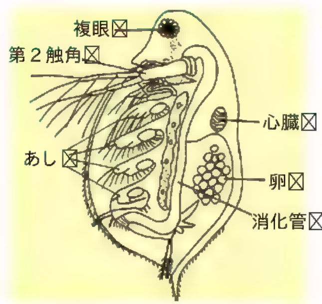


図2　ミジンコ

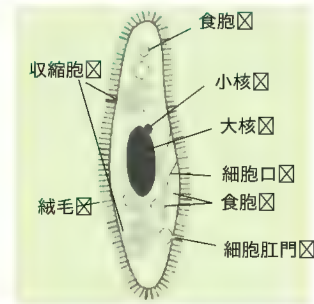


図3　ゾウリムシ

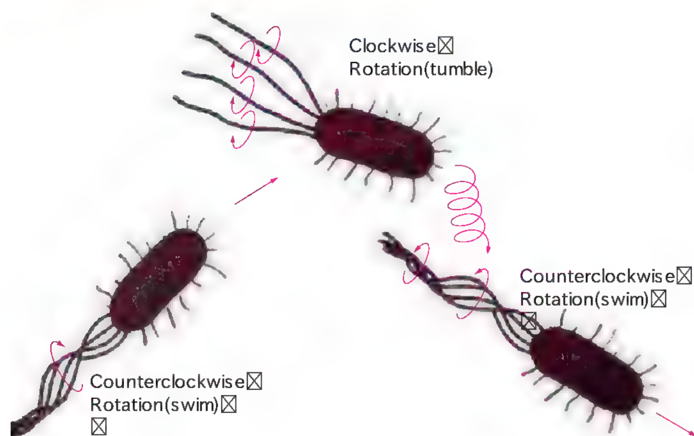


図4　バクテリアの泳ぎかた

をホウネンエビだとすると，一般的に組み込みOSは，上の例ではゾウリムシからバクテリアに対応する．組み込みOSのアーキテクチャを決める主要な要因は，ハードウェア制約と時間制約だろう．このほか，コストやセキュリティなどもかかわってくる．

Windowsをすべての組み込みシステムで使えない理由注7を理解するためには，時間制約を例にするとわかりやすい．人間相手のPCと異なり，実世界の事象を相手にする組み込みシステムでは，一般に厳しい時間制約を持っている場合が多い．いい換えると，扱うべき時間の単位が小さいことと，ばらつきが少ないことを求められるからである．ゾウリムシのサイズでミジンコ型の推進装置を使っても前に進めないように，ms単位で変化する事象を秒単位で観察しても意味がない．

つまり，組み込みで使うためにはWindowsアプリケーションに与えられる時間的な精度は不十分なところがあるということだ．ドライバ・レベルでは組み込みなみの時間精度を得られるが，それは一般にはWindowsを使用しているとはいわず，Windowsを改良したといういいかたになる．PCで使用するCPUは，組み込みで使用するCPUに比べれば数十倍高速である．数GHzのCPUで数μs実行すれば数万ステップを実行できる．PC/Windows環境の実行速度は，組み込みシステムに比較して決して遅くはない（組み込みで使用するクラスのCPUでWindowsを走らせればかなり遅いとは思う）．時間的な精度とは，実行時間の速さだけではない．組み込みシステムでは，実行時間を予測したり解析できることが重要である．WindowsやUNIX，Linuxではこれが難しいのである．

組み込みシステムでは，時間的制約がアプリケーションによって秒単位からms，μsまで広範囲にわたるので，いろいろなオプションが必要になってくる．扱わなければならない時間の単位がmsの場合，秒単位の時間精度しかアプリケーション・プログラムに提供できないOSは利用できない．表2にむりやりOSが生きる世界と生物の世界を対比させてみた．μsより短い時間領域では，OSなしあるいはハードウェアのみで実現する．デバイス・ドライバ・レベルのプログラミングに相当する．つまり，組み込みOSでもだめな時間領域が存在する．

組み込みシステムには，たとえば，ハードディスクが使えない，乾電池で動かす，ROMからブートするなどの物理的な制約が発生する．組み込みOSといった場合，これらのハードウェア環境に耐えるOSという意味で使うのが正しいかもしれない．時間制約を実現するためのOSという意味ではリアルタイムOSということばを使う．ただし実際は，組み込みシステムは時間制約を課せられる場合がほとんどなので，今までは組み込みOSとリアルタイムOSをとくに区別しないことが多い．しかし，今後はわからない．ハードウェア制約で表2のようなものを作ることが可能になるかもしれない．

● 機能追加できないのが組み込みOS

PCの場合，ワープロとして利用したりゲームをしたりと，多目的に利用することができる．PCのOSはこういった意味で汎用OSと呼ばれる場合がある．一方，組み込みOSの場合，特定のハードウェア上で特定のアプリケーションを実行するために利用される．PCのように，後から別のアプリケーション・プログラムをインストールすることがない場合が一般的である．これも組み込みOSの特徴の一つである．

注7：著者は別にWindowsに恨みはない．Windowsには毎日お世話になっている．編集部からの依頼は「なぜ組み込み向けOSが使われるのか．幅広く使われているWindowsを，あえて使わない理由を解説してほしい」とあったので対比させた．

ただし，携帯電話のように，Javaアプリケーションを入れ替えて利用できる機能が求められる場合もある．こういった場合，PCと違う点は，追加できるものは特定の機能の範囲内に限られるということ，また電子マネーや個人情報などの重要な情報に関連したモジュールの追加や変更をセキュリティに注意しながら行わなければならないという点である．後者では，どのように情報を保護するかが重要になってくる．同一のJavaアプリケーションを再ロードすると，ほかのJavaアプリが使用しているスタックが丸見えになるというバグが発覚し，回収された携帯電話がある．発売してから1～2週間後，動作がおかしいことがわかり，あるマスコミがある国立の研究所に調査を依頼して確認し，メーカが回収を始めたのがほぼ1か月後で，この間に約70万台が出荷されていた．端末を無償交換するだけでなく，個人情報が漏れたことによる2次的被害についてもメーカが補償するとすれば，かなり厳しい状況になったと思われる．また，個人情報が漏れたことに対する償いとして何をすればよいのかがこれからの課題である．

組み込みOSとは別の次元の話だが，このバグの原因はかなり初歩的で，ソース・コード・レビューかテスト・プロセスがしっかりしていれば防げたのではないかと業界ではうわさされている．メモリ・アクセスに関する保護機能をもった組み込みOSであれば，あるアプリケーションが別のアプリケーションのデータに不正にアクセスしようとすると，アクセスしようとしたアプリケーションのみがアボートされる．そのままアクセスされることはないし，アクセスされた結果，システム全体が整合性を失ってOSもろともダウンすることもない．OSもろともダウンすると証拠が残らないので，悪意を持った人にはつごうが良く，デバッグする人にはつごうが悪い．

しかし，一度出荷してしまうと，時間との戦いになる．コンシューマ製品の場合，回収を決断することが遅れるとその間に被害は指数関数的に増大してしまう．初期対応を誤ると社会的信用を失う．ミッション・クリティカルな組み込み機器の開発は真剣勝負である．

このようにユーザがアプリケーションを入れ替えることを前提としたセキュリティが求められる場合もあるが，その一方で，計量関係の機器や医療機器ではアプリケーションを勝手に変更できないことが求められる．このあたりが組み込みOSをひとくくりにできない理由である．お肉屋さんのはかりなどの法定計量機器では，型式承認が必要であり，出荷後ROMを差し替えたりするとわかるようなしくみがとられていたが，最近ではフラッシュ・メモリが利用されるので，ソフトウェアでどのように型式承認を維持するかが課題になっている．タクシーに乗るとメータの調節つまみところが鉛で封印されているが，ソフトウェアにどのように封印を付けるかということである．

高機能化が進み，開発期間が短くなっている昨今では，役所の承認手続きなど待っていられないのが現状である．また，バージョン・アップのたびに申請されたら役所も人を減らせない．「最近肉の量が少ない」とが「多い気がする」程度ならお笑いネタだが，医療機器を操作して記録が残らなければ完全殺人ミステリ小説に加担することになる．セーフティ・クリティカルな機器を開発する組み込みエンジニアは，清く正しくなければならない．

● コストが重要

日本の組み込み機器の特徴は民生品中心であること，そして，ほとんどの場合量産品であることが特徴である．欧米のように軍事産業がけん引役ではないので，コストを度外視して性能を追及するわけにはいかない．

組み込みOSを使用する目的は，短期開発を実現するためとか，リアルタイム性を簡単に実現するためとかいろいろある．コストはそれぞれの目的に対して評価する必要がある．組み込みOSを選択する際に考慮すべき点については，参考文献(6)にまとめた．

組み込みOSの概要

● 組み込みOSの種類

現在の組み込みOSの歴史は，マイクロプロセッサとともに始まった．8080とかZ80のころである．当時半導体メーカが提供した組み込みOSには，IntelのiRMXとか，MotorolaのRMS68Kなどがある．半導体メーカ以外では，Programming TechnologyのMTOSがあった．MTOS自身はもうなくなったが，そこから派生したOSが今でも使用されている．

半導体メーカ以外のOSで，そのころから健在なのは6809の9を取って命名されたOS-9がある．最近はRTOSメーカの数は買収などで減少している．それでも商用RTOSを世界中で数えたら100以上はあると思う．The RTOS buyer's guide(7)を見ると，約60件が紹介されている．1997年ごろにここからダウンロードした資料では100件近く掲載されていた．新しく追加されたRTOSもあるので姿を消したRTOSは単純に40件とはいえない．Comp.realtime: A list of commercial real-time operating systems8には，現在約40件が紹介されている．昔は90件も紹

表2　OSが生きる世界との対比

生　物	大きさ	移動方法	何で見るか	OS	時間制約の粒度
ホウネンエビ	2cm	足	肉眼	Windows	1s
ゾウリムシ	0.1mm	繊毛	虫眼鏡	組み込みLinux	数ms
コレラ菌	1μm	繊毛の回転	光学顕微鏡	RTOS	数μs注8
インフルエンザ・ウイルス	100nm	風まかせ	電子顕微鏡	System LSI	ns

注8：RTOSにはスレッド型とプロセス型がある．この値はスレッド型である．プロセス型の場合は数十μsになる．

介されていた．

今回気付いたことは，どちらのサイトにもイーソルのPrKernelが紹介されていた．日本製のRTOSが海外で紹介されていることは珍しい．日本にITRONがあることは知られているが，海外ではeCOSのITRONインタフェースか，U S SoftwareのTronTask!として知られているのみであったが，ようやく日本製ITRONが紹介されるようになった．

このほかの組み込みOSの紹介ページとしてはRTOS at a Glance[9]，国内では「リアルタイムOS情報[10]」がある．これらのサイトを見ればわかるように，これだけの数の組み込みOSが存在している．これらをまとめて概要を述べるのは無謀だ．

とはいっても，共通する部分もあり，同じような機能を提供している部分もある．また，機能やインタフェースを統一しようとする動きもある．ITRON[11]やIEEEのMOSI（Microprocessor Operating Systems Interfaces）[12]，POSIX 1003.1b（Portable Operating System Interface for UNIX）[13]などである．自動車に特化したものとしてOSEK（Offene Systeme und deren Schnittstellen fur die Elektronik im Kraftfahrzeug）[14]がある．

したがって，組み込みシステムで使用されるOSの概要を知りたい場合，これらの資料を読んだり，せめて目次を見ておくなどすると先輩たちの話に出てくる単語を知ることができる．ちなみにMOSIは英語で260ページ，POSIXはさらに膨大である．POSIXはUNIXベースなので，MOSIかITRONの仕様構成に目を通すと良いかもしれない．もっとも，実際に読む必要がある人は組み込みOSを作る人である．

MOSIで記述されている機能は，メモリ管理，時間管理，データ管理，プロセス管理，プロセスの同期と通信，環境，例外処理である．ITRON4.0では，タスク管理，タスク付属同期機能，タスク例外処理機能，同期・通信機能，拡張同期・通信機能，メモリ・プール管理機能，時間管理機能，システム状態管理機能，割り込み管理機能，サービス・コール管理機能，システム構成管理機能になっている．ITRON4.0は日本語で370ページなのでMOSIよりも少し仕様が膨らんでいる．

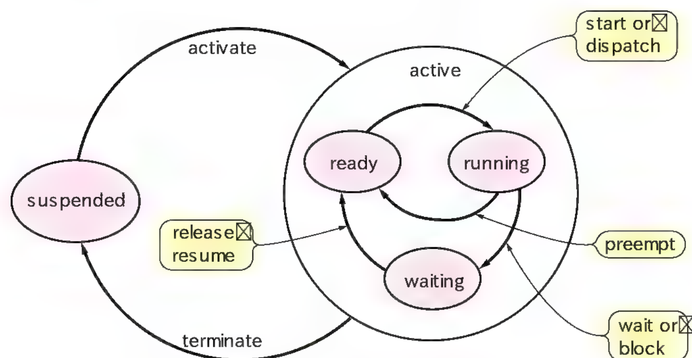


図5　一般的なタスクの状態遷移

● マルチタスク機能

組み込みシステムにほぼ共通していえることは，複数のタスクを並行して走らせる機能である．この機能は，タスクという概念を作り出し，そのタスクを起動したり止めたりする．あるタスクが止まると別のタスクが動き出す．組み込みシステムの中では必ず一つのタスクだけが実行されている．タスクを切り替える機能はカーネルの中のディスパッチャと呼ばれるモジュールの中に実装されている．どのタスクを走らせれば良いか決めるのはカーネルの中のスケジューラと呼ばれる部分である．図5に一般的なタスクの状態遷移を示す．実際は各種の組み込みOSによって名前の付けかたや状態間の関係などが変わる．アクティブ状態の中に，ready，running，waitingの三状態がある．これらの状態間を遷移させる操作のうちdispatch，preempt，block，resumeはOSによって行われる．そのほかの操作は，システム・コールによってタスク自身が行う．

組み込みOSのスケジューリングは単純である．各タスクにはプライオリティ（優先度）が割り当ててあり，実行可能なタスク（Ready状態のタスク）の中でプライオリティの高いタスクがCPUを占有して実行する．同一プライオリティのタスクが複数ある場合は，先に実行可能になったほうがCPUを占有し続ける．ルールはこの二つだけである．プライオリティの高いタスクが待ち状態などに遷移しない限り，プライオリティの低いタスクは実行されない．現在実行中のタスクよりもプライオリティの高いタスクが実行可能になると，今まで実行していたタスクが実行状態（Running状態）ではなくなり，新しい実行可能になった高プライオリティのタスクがすぐに実行状態になる．この機能をプリエンプションという．CPUを効率的に利用するために有効な機能である．プリエンプションを使用できないスケジューラの場合，優先度の高いタスクはただちに実行状態になれずに，プライオリティの低いタスクが制御を放すまで実行可能状態のままになる．

同一プライオリティのタスクが複数あった場合，一つのタスクが実行し続けるのは不合理な気もする．その場合は，ラウンド・ロビン・スケジュールを行う．ラウンド・ロビン（round robin）は，各タスクに持ち時間を与えて与えられた時間を使い切ったら，次に実行可能になったタスクに実行権を渡して，今まで実行していたタスクはそのプライオリティの一番後に回るスケジューリングである．ロビンフッドのように，ひとり勝ちしている金持ちから金を奪い，貧乏している人に施すスケジューリングという意味かもしれない．また，ツグミ（robin[注9]）のようにエサをついばみながらうろうろ歩き回るようなスケジューリングという説もある．筆者の家のあたりにも冬になるとツグミがやってくる．鳥のくせに飛んでいるところを見たことがな

注9：Japanese robinはコマドリ，日本のツグミによく似ているのはscrub robinでこれはいつもやぶの中にいるらしい．Round robinのrobinがコマドリかツグミかは確認できなかった．

い．朝会社に行く途中ですれ違うという不思議な鳥である．

プライオリティが同一であればラウンド・ロビンで救われるが，プライオリティが低いタスクはどうするのか．プライオリティ値が一つでも低いと実行できなくても良いのか，という問題がある．それで，組み込みOSの中にもReady状態になってからの経過時間に応じてプライオリティを上げるスケジューラ・オプションを持っているものもある．これらの機能は，CPU時間を公平に分配する機能だが，動作が確率的になり，予測しにくいので，リアルタイム性を要求される組み込みシステムでは使えない．リアルタイム・システムは外部からの働きかけに対して，定められた一定時間内（デッド・ライン）に応答しなければならない．デッド・ラインを保証できることが，すなわちリアルタイム・システムである．ラウンド・ロビンの下では外部からの働きかけとはまったく独立にタスクが止められてしまう．いつ止められるかはラウンド・ロビン・スケジューラが使っているインターバル・タイマの精度の範囲で確率的に変動してしまうので，リアルタイム性が損なわれることになる．しかも一度止められると，同一プライオリティで待っているタスクについて処理が一巡するまで動けなくなってしまう．必要なときに必要なタスクが動けるようにタスク間の同期構造を設計していないと，「サイコロを振ってどのタスクが動くか決める」ような実装になってしまう．リアルタイム・システムにとってはロシアン・ルーレット[注10]のようなものである．この意味で，これらの機能を利用することはタイミング設計の敗北であり，とくに開発の後半になって導入することは避けたいが，実際には利用してしまう事例が多い．

リアルタイム性

リアルタイム・システムと非リアルタイム・システムの違いは，表3のようにまとめることができる．

単純に実行時間が速いだけなら，数GHzで動くCPUを積んだPCのほうが，せいぜい数百MHzのCPUで動作する組み込みシステムよりも，リアルタイム処理向きということになるが，それは違う．平均実行時間が速くてもときどき非常に遅くなるときがある場合，リアルタイム性があるとはいえない．

コンピュータで行う処理にはイベント・ドリブンのリアルタイム処理と，一括で行うバッチ処理がある．銀行の勘定系処理でいうと，ATMから現金を引き出す処理がリアルタイム処理である．もう一つは企業間の口座振替処理でセンタ・カットと呼ばれているものがある．電話会社とかクレジット会社が必要な取引データを書き込んだファイルを銀行の計算センタに送って，計算センタで預金者の口座から一括で振り替えを行う．ATM利用のピーク時には1秒間に1,000件程度，センタ・カットは多い日で一日1,000万件を超えるそうである．どちらの処理も膨大だが，ATMのほうは一件ずつにデッド・ラインがあるのに対して，センタ・カットのほうは全体をまとめて指定された期日までに処理すればよい点が異なる．スループットで評価されるのはセンタ・カットである．組み込みOSに要求されるのはATM型の処理である．

表3 リアルタイム性の特徴

	非リアルタイム・システム（おもに時間分割システム）	リアルタイム・システム
処理能力	スループットで評価	スケジュール可能性で評価
応答性	平均応答時間の速さで評価	最悪応答時間の短さで評価
過負荷状態	公平であること	安定であること

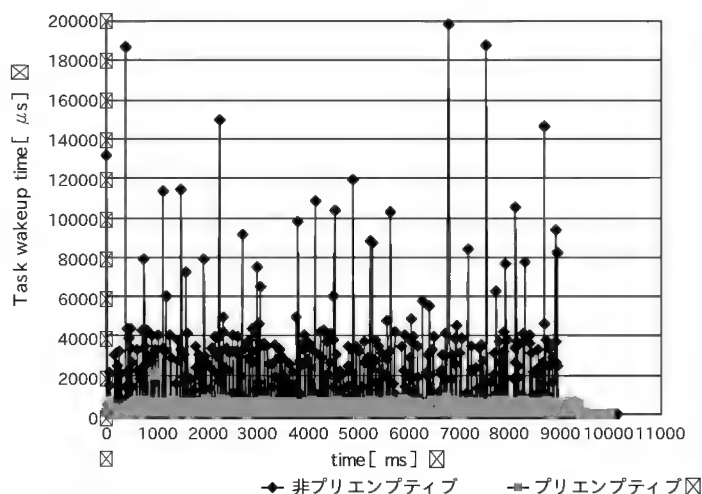


図6 プリエンプションの効果[15]

図6は，参考文献15)に掲載されていたLinuxをプリエンプション動作させた場合と非プリエンプション動作させた場合のタスク起床時間の測定データである．横軸は測定時間を表している．測定はファイル・アクセスによる負荷をかけて行われた．

非プリエンプションの場合のように，実行時間がばらついていてはリアルタイム・システムには使えない．動作が確率的になるというのはこの意味である．つまり，「速いときもあれば遅いときもある」ような，やってみなければわからないというシステムはリアルタイム・システムではない．そのときの状況で，速く効いたり，なかなか効いてこないブレーキの付いた自動車では怖くて運転できない．

最短起床時間は，非プリエンプションの場合が18μsでプリエンプションの場合は24μsである．つまり，どちらが速いかと問われれば非プリエンプションのほうが速いといえなくもない．しかし，リアルタイム・システムで使用できるOSはプリエンプションOSのほうである．このことを説明するために起床時間の分布グラフを作成した．

図7と図8は，起床時間の度数分布を示している．どちらも

注10：ロシア式ルーレット（Russian roulette）．かけの一種．リボルバに一発の弾丸を入れ，各人が順番に銃口を頭に向けて引き金を引く．

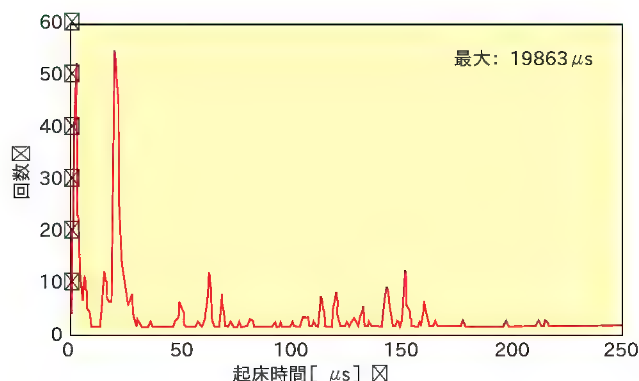


図7　非プリエンプション時の起床時間分布

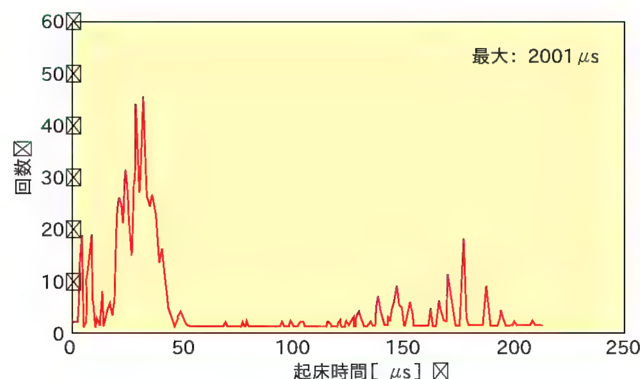


図8　プリエンプション時の起床時間分布

横軸は起床時間である．測定ではDMA転送によって負荷をかけているが，おそらく50μs以下の起床時間はリソース競合が起こらないタイミングの起床時間の分布を反映している．ここの分布を比較すると，非プリエンプションでは20μsあたりがピークであり，プリエンプションでは30μsあたりがピークになっている．プリエンプションを行わないほうが，短い時間で起床することがわかる．負荷をかけない状態では非プリエンプションのほうが有利だが，負荷がかかると非プリエンプションは非常にもろいことがわかる．このデータは組み込みLinuxのものだが，本当のRTOSを使えばこれほどばらつくことはないだろう．プリエンプションの場合でも，最長で2msかかっているが，一般的なRTOSではタスクの起床に2msかかるというのは考えられない．今の組み込みLinuxはもう少しリアルタイム性が向上していると思う．

プリエンプションを行うことでOSの仕事が増える分だけオーバ・ヘッドも増えるが，それ以上にCPUの利用効率が向上するので実質的にアプリケーションの実行効率が良くなり，高負荷時の安定度も増す．PCに比べればプアなCPUを利用する組み込みシステムでは，CPUを効率よく利用することが求められる．この意味でも，プリエンプションはつごうが良い．

おわりに

数年前から組み込みに注目する人が増えてきている，その割には「Interface」の発行部数が増えた話は聞かない．という話は別として，大人はすぐに，組み込みとは何かと定義したがる．しかし，子供はそんなことは気にしない．目を輝かせて珍しいものに手を出す．よけいな回り道はしない．

毎日の生活の中で，何をしたいか，何があったら便利か，最近何に感動したか，それはなぜか，ということについてまず考えてみる．次に，どのように実現したらよいか，どのようにすれば簡単に実現できるかを考えてみる．実現する手段のひとつが組み込みシステムである．その組み込みシステムを実現する手段のひとつが組み込みOSである．組み込み業界に新しく入る人たちは，組み込みOSを既存のOSと比較して定義する必要はない．既存のOSから出発して，あれば便利な機能は追加すればよい，不要な機能は削除すればよい．毎日変化しているのが組み込みOSである．

自動車のように複数のCPUがネットワークでつながっている組み込みシステムでは，ネットワーク上に分散して協調しながらリアルタイムで動くOSが必要になってくる．それはどのようなOSなのかを考える際に組み込みOSの定義などは不要である．多機能のICカードでは，プログラムの入れ替えとセキュリティが重要で，タスクの切り替えなど必要としない．モニタ部分がなくて，デバイス・ドライバとミドルウェアの構成になるかもしれない．はたしてOSと呼んで良いのかさえわからない．つまり，変化の度合いはOSという概念さえ古いものにしてしまうかもしれないのである．

参考文献

(1) システム検証の科学技術，http://unit.aist.go.jp/informatics/verification2004/
(2) 独立行政法人 情報処理推進機構；http://www.ipa.go.jp/
(3) ポスト PC時代のOS研究，情報処理，vol.41, no. 10, 2000.
(4) http://www.palm.com/home.html，http://www.palmone.com/jp/
(5) http://www.symbian.com/Japan/
(6) 藤倉 俊幸；リアルタイムOSを使う理由，TECH I 実践リアルタイムOS活用技法，vol.19, pp.4-6.
(7) http://www.omimo.be/encyc/buyersguide/rtos/Dir228.html
(8) http://www.faqs.org/faqs/realtime-computing/list/
(9) http://www.onesmartclick.com/rtos/rtos.html
(10) http://www.sanritz.co.jp/realtime/
(11) http://www.assoc.tron.org/itron/
(12) IEEE Std 855-1990
(13) David R. Butenhof；Programming With Posix Threads, Addison-Wesley, 1997.
(14) http://www.osek-vdx.org/
http://www.ertl.ics.tut.ac.jp/~kaz/research/osek/osek_contents.html に翻訳が公開されている．
(15) 宍田 啓樹；ソフトリアルタイムシステムをリアルタイムLinuxに移行するプロセス，TECH I 組み込みLinux入門，vol.16, pp.218-229.

ふじくら としゆき　組み込みコンサルタント
tfujikura@jcom.home.ne.jp

3. 動かない！を動かすための
実例で学ぶ ハードウェアのデバッグ

川本 泰久

設計したハードウェアがすんなりと動くことは少ない．実際には，基板製造業者から到着した試作基板に電源を投入しても，ウンともスンともいわないことが多いのではないだろうか．ハードウェアの開発は設計までが半分であり，それをデバッグして動かすまでの仕事がまだ残っている．

本章では，実際に筆者が開発した基板をもとに，遭遇したバグを一つずつ修正し，製品出荷にいたるまでの過程を解説することにより，ハードウェアのデバッグ技法を解説する．

（編集部）

本章では弊社（デバイスドライバーズ）で販売している組み込みボード「E!Kit-1100」を開発するにあたり，実際に行ったハードウェアのデバッグについて解説します．

ハードウェアのデバッグの目的は，ハードウェアを安定して動作させることです．目の前にハードウェアがあるとすぐに動作させてみたくなるものですが，デバッグを行う前にまず基本部分の確認を行います．この確認をおこたって問題があった場合，一見動作しているようにみえても安定した動作は望めません．今後のデバッグにも影響します．急がば回れです．

今回用いたE!Kit-1100の概要

● 学習・実験用途向け組み込みボードの概要

「E!Kit-1100」は写真1のような，クレジット・カード2枚と同じサイズ（110×90mm）のボードに，400MHzで動作するAMD社 Alchemy Au1100 CPU，128Mバイト SDRAM，8Mバイト・フラッシュ・メモリを搭載し，100Base-TX LAN，USBホスト/Targetコントローラ，CF，シリアル・ポート，JTAG，LCDコントローラなどの機能を提供する，学習・実験用途向け組み込みボードです．おもな仕様を表1に示します．

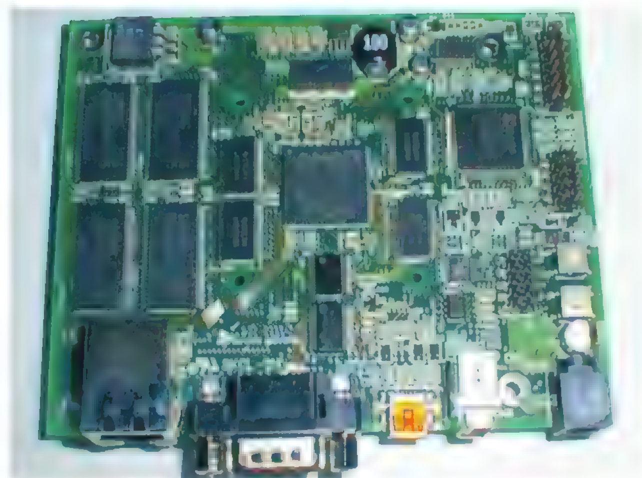


写真1　学習・実験用途向け組み込みボード E!Kit-1100

表1　学習・実験用途向けボード E!Kit-1100のおもな仕様

CPU	AMD Au1100-400MHz，32Kバイト・キャッシュ，MIPS32アーキテクチャ，リトル・エンディアン
SDRAM	128Mバイト×32ビット
フラッシュ・メモリ	8Mバイト×16ビット
シリアル	UART×1（9ピンDサブ，ハードウェア・フロー制御対応）
Ethernet	10/100Base，Am79C874 PHY
USB	Type-A×1，Type-mini AB×1（ホスト側とターゲット側をサポート）
CF	Type-I/II×2（Microdrive，無線LAN対応）
Expansion	100pin×2 BUS：32ビット・データ/32ビット・アドレス/チップ・セレクト×2 SD：×2 LCD：STN/TFT IrDA：SIR/MIR/FIR UART：×2 GPIO：Au1100×18（一部他の信号と切り替え）/CPLD×6
CPLD	Xilinx XC95144XL 100ピン 144マクロセル 3300ゲート中の約80%をシステムでCF制御用に使用
EJTAG	14ピン
JTAG	10ピン
LED	Standby，Main，Core power（on Board） Ethernet Tx and Rx（on RJ45）
スイッチ	Power，Reset（on Board）
CPU温度特性	動作時：0℃～50℃（周辺温度）
電源	+5V±5%，max 3.2A
サイズ	110×90mm
OS	Linux 2.4系カーネル
価格	定価¥98,000（本体ボード，電源，ソフトウェア開発環境付き），（Expansion機能は別基板に実装するため含まれない）

● Au1100 プロセッサについて

「E!Kit-1100」で採用したAMD Au1100プロセッサは，400MHz動作時で平均250mWという超低消費電力を実現する，MIPS32テクノロジをベースにしたSoC（System On a Chip）プロセッサです．1チップにCPUコア，メモリ・コントローラのほか，10/100Base Ethernetコントローラ，USBデバイスとホスト，PCMCIA/コンパクト・フラッシュ・インターフェース，IrDA，AC-97インターフェース（外部CODECが必要），SDカード・コントローラ（二つ），UART（三つ），SSIコントローラ（二つ），JTAG，GPIOピン，最大800×600ドット（SVGA）を表示できるLCDコントローラを搭載しているのが特徴です．詳細仕様やデータ・ブックはAMDのサイトから入手可能です．

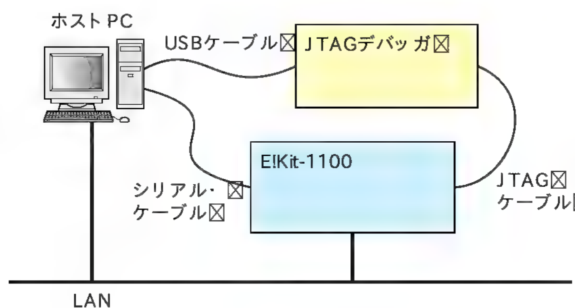


図1　ホスト PC，JTAGデバッガとボードの接続方法

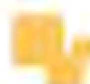


ハードウェアのデバッグとは

● デバッグの基本

まず，各種電源の電圧が規定の範囲にあるかを，テスタなどで確認します．電解コンデンサなどは極性の確認も行います．次にクロックなどの波形や周波数をオシロスコープなどで確認しましょう．

ハードウェアの動作確認を行うためには，ソフトウェアを実行し，各種機能を動作させる必要があります．しかし，開発途中のハードウェアではモニタ（パソコンでBIOSに相当する）でさえ動作していません．それでは何もできないので，JTAGデバッガを使用してデバッグを進めます．



ディスプレイやキーボードのない環境でのデバッグ

組み込み系のボードの場合，ディスプレイやキーボードがないことが多く，特殊なデバッグ方法があるのではと思う人も多いと思います．しかし，ディスプレイやキーボードがあるボードでも，初期の段階では動作していないこともあり，実はさほど違いはありません．

まずホスト PC，JTAGデバッガとボードを図1のように接続して，モニタを動作させるのに最低限必要な機能（CPU，メモ

リスト1　バッチ・ファイルの例――ハードウェアの初期設定

```
;
; BATCHFILE for TRACE32 which initializes E!Kit-1100
; Device Drivers Limited        ←「；」で始まる行はコメント行
; 02/19/04
;

;
; TRACE32
;
 RESET
 SYSTEM.JTAGCLOCK 10000000.     } JTAGデバッガ設定
; SYSTEM.OPTION TURBO ON
 SYSTEM.UP
 SYSTEM.RESETOUT

;
; Au1100(SDRAM)
;
 DATA.SET 0xB4000000 %LONG 0x005922A9
 DATA.SET 0xB4000004 %LONG 0x005922A9
 DATA.SET 0xB400000C %LONG 0x001003F0
 DATA.SET 0xB4000010 %LONG 0x001043F0    } Au1100 SDRAM
 DATA.SET 0xB4000018 %LONG 0x6400061A      コントローラ設定
 DATA.SET 0xB4000024 %LONG 0x00000023
 DATA.SET 0xB4000028 %LONG 0x00000023
 DATA.SET 0xB4000018 %LONG 0x6600061A

;
; Au1100(STATIC)
;
 DATA.SET 0xB4001000 %LONG 0x00060043
 DATA.SET 0xB4001004 %LONG 0x040181D7
 DATA.SET 0xB4001008 %LONG 0x11F83FE0
 DATA.SET 0xB4001010 %LONG 0x000000C0
 DATA.SET 0xB4001014 %LONG 0x22080A20
 DATA.SET 0xB4001018 %LONG 0x11803F80    } Au1100スタティック・
 DATA.SET 0xB4001020 %LONG 0x00000080      バス・コントローラ設定
 DATA.SET 0xB4001024 %LONG 0x22080A20
 DATA.SET 0xB4001028 %LONG 0x11A03F80
 DATA.SET 0xB4001030 %LONG 0x00000002
 DATA.SET 0xB4001034 %LONG 0x280E3E07
 DATA.SET 0xB4001038 %LONG 0x10000000

;
; Au1100(CLOCK)
;
 DATA.SET 0xB1900014 %LONG 0x00000100
 DATA.SET 0xB1900020 %LONG 0x00300000
 DATA.SET 0xB1900028 %LONG 0x0A000000    } Au1100クロック設定
 DATA.SET 0xB1900060 %LONG 0x00000021
 DATA.SET 0xB1900064 %LONG 0x00000008

;
; Au1100(SYSTEM BASE)
;                                         } Au1100ピン設定
 DATA.SET 0xB190002C %LONG 0x000005A0

;
; CPLD
;                                         } 周辺回路設定
 DATA.SET 0xB8000010 %LONG 0x00000000

;
; FLASH
;
 FLASH.RESET       ← JTAGデバッガのフラッシュ・メモリ定義
 FLASH.CREATE 1. 0xBF800000--0xBFFFFFFF 0x10000 AM29LV100 WORD

 ENDDO
```

B::DATA.DUMP 0xA0000000--0xA000FFFF /LONG

address	0	4	8	C	0123456789ABCDEF
D:A0000000	DFFF87EF	21FFFFF7	FBFDBFBD	7AFFF3DF	[illegible]
D:A0000010	6FDFFB6B	976FF9F8	FFBDFFDF	9FFCBFF7	[illegible]
D:A0000020	7D4FF3F7	9FFF9B06	FBDFDFFF	7FFDB32F	[illegible]
D:A0000030	BFFEDFEB	DFDF17BB	6FFD3F7B	BEDFFBFE	[illegible]
D:A0000040	B5FFFBA7	B7FF7ED7	9E3FFBF3	BEFAF714	[illegible]
D:A0000050	EDDFEFBF	AFEBFEF3	FFFFBF2F	DBF7BCFF	[illegible]
D:A0000060	BBDEFFFB	FFD73EE7	7FFEFFE7	FF9FFFBD	[illegible]
D:A0000070	FDE6D0E3	B2ED79FE	EFAFFFDF	BDFF3F3B	[illegible]
D:A0000080	FBDE3BFD	E7EFE786	EBFFF68F	ECF9B7FD	[illegible]
D:A0000090	F7F37FEB	BFD7BFBF	F9BFBFEF	7FFA7BDE	[illegible]
D:A00000A0	D6BF37EF	BFDF97DE	BFFC97FB	E73FBF3E	[illegible]
D:A00000B0	F7F97FF1	BFFFFAF2	7BF7FDF3	FFEEFF7F	[illegible]
D:A00000C0	CFFDCFBF	E3EB3F6F	37FFAF6B	7DCFF5F7	[illegible]
D:A00000D0	74CAF3FA	A7DFF7D1	7BFDBF73	7FED740F	[illegible]
D:A00000E0	FDFEF7FD	AF8DF77C	D73A3959	FFFFFEBF	[illegible]
D:A00000F0	1BDEFF8E	7DFDFFDD	3FFE7FFB	7A6E3F7B	[illegible]
D:A0000100	5B7F7EEB	BFFB3E5B	DBBDFBF9	7FFD7FFE	[illegible]
D:A0000110	07DEBC91	D7F67EFD	FFEFF15A	7EFFAF37	[illegible]

図2　メイン・メモリの内容

B::DATA.DUMP 0xA0000000--0xA000FFFF /LONG

address	0	4	8	C	0123456789ABCDEF
D:A0000000	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A0000010	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A0000020	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A0000030	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A0000040	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A0000050	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A0000060	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A0000070	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A0000080	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A0000090	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A00000A0	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A00000B0	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A00000C0	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A00000D0	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A00000E0	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A00000F0	0055AAFF	FFAA5500	0055AAFF	FFAA5500	[illegible]
D:A0000100	5B7F7EEB	BFFB3E5B	DBBDFBF9	7FFD7FFE	[illegible]
D:A0000110	07DEBC91	B7F67EFD	FFEFF15A	7EFFAF37	[illegible]

図3　ウィンドウの内容が書き換わる

リなど)の動作確認を行います．

● ハードウェアの初期設定

通常は，モニタの初期化コードでハードウェアの初期設定が行われますが，まだモニタも動作していない状態では，JTAGデバッガを使用して必要最低限の初期設定を行います．初期設定のためのコマンドを一つ一つ手で入力してもかまいませんが，まちがいも発生しやすく，手間もかかります．JTAGデバッガにはコマンドをバッチ・ファイルとして記述する機能があるので，バッチ・ファイルをあらかじめ作成し，毎回使用したほうがまちがいも減るうえに楽です．バッチ・ファイルの例をリスト1に掲載します．

バッチ・ファイルにはJTAGデバッガの設定，CPUや周辺回路の初期設定，フラッシュ・メモリの定義が記述されています．フラッシュ・メモリの定義は，JTAGデバッガがフラッシュ・メモリの消去や書き込みを行うのに必要な情報(アドレスや種類，セクタ長)を定義するためのものです．

● メイン・メモリの動作確認

ハードウェアの初期設定が終わったら，まずメイン・メモリの動作を確認します．JTAGデバッガでメイン・メモリをリード/ライトすることで確認できます．

```
DATA.DUMP 0xA0000000--0xA000FFFF /LONG
```

まず上記のdata.dumpコマンドでメイン・メモリの一部を表示します．実行すると図2のようなウィンドウが開き，メイン・メモリの内容が表示されます．

```
DATA.SET 0xA0000000--0xA00000FF
                          %LONG 0x0055AAFF 0xFFAA5500
```

次に上記data.setコマンドでメイン・メモリの内容を変更します．実行すると図3のように先ほど表示されていたウィンドウの内容が書き換わります．

この結果から，メイン・メモリに対して正常にリード/ライトできていることがわかります．もし動作に不具合があれば，以下のように原因をみつけて対処します．

- メイン・メモリの内容が変更されない
  →ライト・イネーブル信号に問題がある
- メイン・メモリで特定のビットが"1"もしくは"0"に固定される
  →データ・バスに問題がある
- メイン・メモリの特定のアドレスしか内容を変更していないにも関わらず，複数のアドレスの内容が変更される
  →アドレス・バスに問題がある

問題がありそうな信号が接続されているかどうかはテスタで導通を確認し，信号が別の信号に短絡しているかどうかはオシロスコープで信号レベルを確認します(信号が短絡している場合，"H"でも"L"でもない電圧レベルになる場合がある)．短絡が確認できた場合はテスタを使用して短絡箇所を探しましょう．

信号が接続されていない場合のおもな原因は，以下のものが考えられます．

- はんだ付け不良
- プリント配線板の製造上の問題で配線が切れている
- 回路がまちがっている

はんだ付け不良の場合には，はんだ付けを再度行います．プリント配線板に問題がある場合は線材を使用して配線を行います．写真2(p.59)は配線の例です．緑色の線材を使用して信号を接続しています．

逆に信号が短絡しているおもな原因は以下のものが考えられます．

- はんだブリッジ
- プリント配線板の製造上の問題で短絡している
- 回路がまちがっている

はんだブリッジの場合ははんだ付けを再度行います．プリント配線板に問題がある場合は配線をカットし，線材を使用して正しい配線を行い，対処します．

● フラッシュ・メモリの動作確認

メイン・メモリの次はフラッシュ・メモリの動作を確認します．JTAGデバッガのフラッシュ・メモリ関連のコマンドを使用して確認できます．

```
DATA.DUMP 0xBFC00000--0xBFC0FFFF /LONG
```

まず上のdata.dumpコマンドでフラッシュ・メモリの一部

を表示します．実行すると図4のようなウィンドウが開き，フラッシュ・メモリの内容が表示されます．

```
FLASH.ERASE ALL
```

次に上記flash.eraseコマンドでフラッシュ・メモリの内容を消去します．実行すると図5のように先ほど表示されていたウィンドウの内容がすべて0xFFに書き換わり，コマンドも図6のように正常終了します．

この結果からフラッシュ・メモリが動作していることがわかります．もし動作に不具合があればメイン・メモリ同様，原因を見つけ対処します．

● その他機能の動作確認

ここまで動作するようになれば，実際にモニタをフラッシュ・メモリに書き込み，モニタ上で動作確認を行います．書き込み後，先ほどの図1のようにシリアル・ケーブル，LANを接続して起動します．正常に動作していればホストPCのターミナルに図7のように起動メッセージが表示されます．

起動メッセージが表示されない場合は，シリアル・ポートの信号線に問題がある可能性があります．ここでも同じようにテ

JTAGデバッガ

JTAGとは，本来ボードのテスト用に規格化（IEEE 1149.1）されたものですが，その信号を利用したデバッガがJTAGデバッガです．プロセッサに内蔵されたデバッグ機能とJTAG用信号経由で通信することで，CPU内部のレジスタや外部に接続されたメモリなどに直接アクセスできるようになります．

JTAGデバッガがない時代にも，ICE（In Circuit Emulator）というものがありました．これはボード上のCPUの代わりにボードに接続し，CPUの代わりに動作させCPU内部のレジスタや外部に接続されたメモリなどに直接アクセスするものでした．CPUの代わりに接続して動作させるため信号線の本数も多く，接触不良や，高速なCPUの場合は動作が不安定だったりで苦労した経験があります．

JTAGデバッガで，「E!Kit-1100」を接続した例を写真Aに示します．この写真のようにJTAG用に用意されたコネクタにケーブルを接続して使用します．図AはJTAGデバッガの実行中の画面です．左下のウィンドウにコマンドを入力するか，マウスでメニューをクリックすることで操作します．JTAGデバッガのおもなコマンドを表Aに示します．

E!Kit-1100を製作するにあたり，デバッグに使用するJTAGデバッガを調査して選択しましたが，一般的なMIPS32用のデバッガでは，次のような問題点があって苦労しました．

1) AMD Alchemy CPUを正式サポートしていない
2) 製品で使用する予定だったフラッシュ・メモリの動作保障（または実績）がない
3) コンパイラや統合環境と別の単体販売で適当な価格のものが少ない

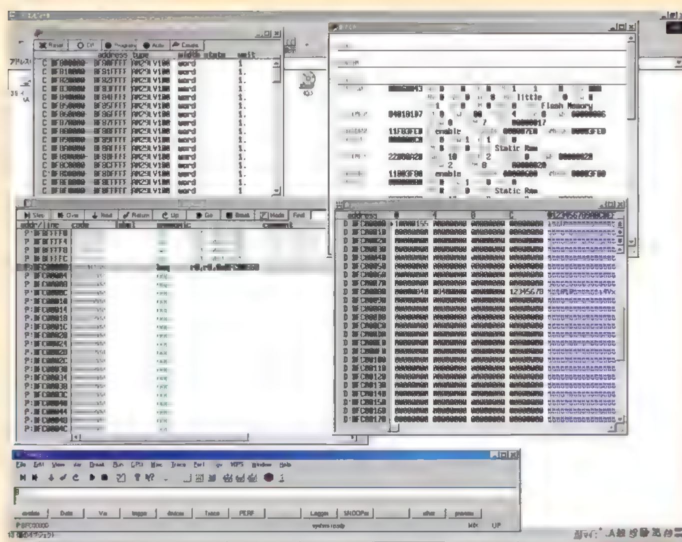
以上のことから，すべての条件を満足する，ローターバッハ製のJTAG Debugger for MIPSを探し出して，製品開発に使用しました．JTAG Debugger for MIPSは，Windows PCからUSB経由で接続できて，ソースコード・レベルのデバッグやトレースもできるものです．弊社では，E!Kit-1100と合わせてJTAG Debugger for MIPSも販売しています．

表A　JTAGデバッガのおもなコマンド

コマンド	動　作
DATA.DUMP	指定したアドレスを表示
DATA.SET	指定したアドレスに指定したデータを書き込む
DATA.TEST	指定したアドレスをテスト
FLASH.ERASE	指定したアドレスのフラッシュ・メモリの消去



写真A　JTAGデバッガ



図A　JTAGデバッガの実行中の画面

New Products ――東芝，Bluetoothを搭載したauの携帯電話「A5504T」を発売
（株）東芝は，Bluetoothを搭載したauの携帯電話「A5504T」を4月中旬に発売する．携帯電話に内蔵のカメラで撮影した写真を，Bluetooth搭載のカラー・プリンタで出力するといった使いかたを想定している．

スタやオシロスコープを使用して信号を確認し，問題があれば対処します．

無事に起動したら環境設定を行います．あとモニタにメイン・メモリとフラッシュ・メモリをテストするコマンドがあるので実行してみました．図8のように正常終了しています．

次に機能の動作確認ですが，モニタの dump，edit コマンドでもある程度デバッグを進めることは可能です．しかし，複雑な操作が必要な場合には，プログラムを作成するのがよいでしょう．ホスト PC 上でクロス・コンパイルを行い，LAN もしくはシリアル・ポート経由でプログラムをロードして実行します．

例として CF の電源制御に使用しているテキサス・インスツルメンツ社製 TPS2223A(Dual-Slot Cardbus Power-Interface Switches for Serial PCMCIA Controllers)をリスト 2のプログラムを実行して動作確認してみます．接続は図9のようになっています．モニタの load，go コマンドを実行した画面が図10です．プログラムを実行しながらオシロスコープで確認します．

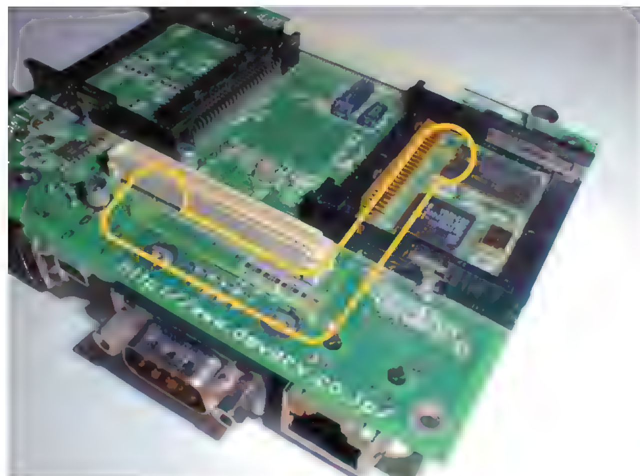

写真2　配線の例



図4　フラッシュ・メモリの内容が表示される

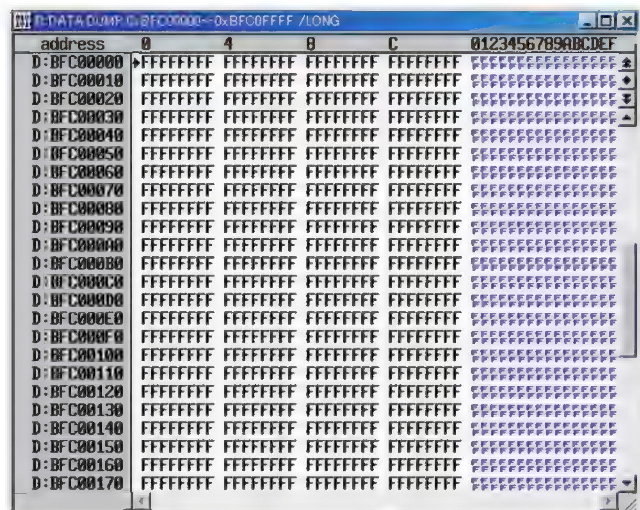


図5　ウィンドウの内容が書き換わったようす

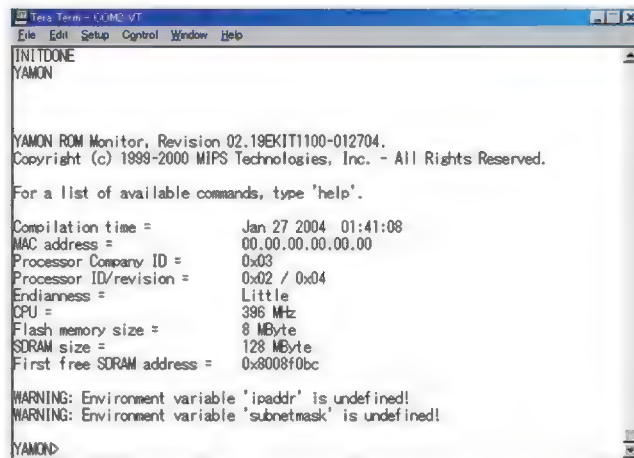


図7　起動メッセージ

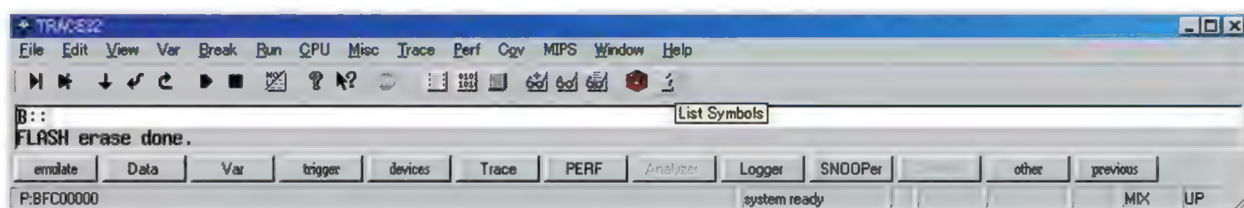


図6　コマンドが正常終了したようす

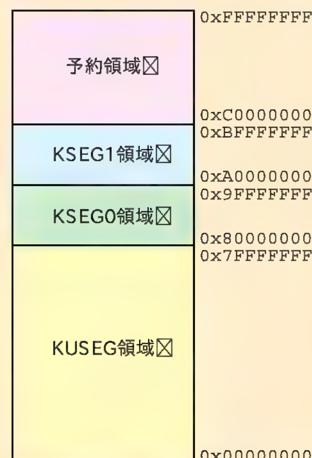
COLUMN 2

MIPSアーキテクチャの論理アドレス空間

MIPSアーキテクチャの論理アドレス空間は図Bのようにいくつかの領域に分かれていて，それぞれ動作が異なります．Au1100の場合は以下のような仕様になっています．

1) KUSEG領域は`0x00000000`から`0x7FFFFFFF`のTLB[注]経由のキャッシュされる2Gバイトのアドレス空間です．ユーザ・モードあるいはカーネル・モードでアクセス可能です．
2) KSEG0領域は`0x80000000`から`0x9FFFFFFF`の物理アドレス空間`0x00000000`から`0x1FFFFFFF`に直接マッピングされた，キャッシュされる512Mバイトのアドレス空間です．カーネル・モードでのみアクセス可能です．
3) KSEG1領域は`0xA0000000`から`0xBFFFFFFF`の物理アドレス空間`0x00000000`から`0x1FFFFFFF`に直接マッピングされた，キャッシュされない512Mバイトのアドレス空間です．カーネル・モードでのみアクセス可能です．

ハードウェアのデバッグではアドレス変換されたりキャッシュされたりすると不都合があるので，通常KSEG1領域を使用して物理アドレス空間にアクセスします．



図B　論理アドレス空間

注：TLB（translation look-a-side buffer）：アドレス変換バッファのこと

```
Copyright (c) 1999-2000 MIPS Technologies, Inc. - All Rights Reserved.

For a list of available commands, type 'help'.

Compilation time =              Jan 27 2004  01:41:08
MAC address =                   00.0e.6c.00.00.00
Processor Company ID =          0x03
Processor ID/revision =         0x02 / 0x04
Endianness =                    Little
CPU =                           396 MHz
Flash memory size =             8 MByte
SDRAM size =                    128 MByte
First free SDRAM address =      0x8008f0bc

YAMON> test
Testing ALL
Testing RAM
Memory test from 0xA008F100 to 0xA7FFFFFC, 10 loops.
Press Ctrl-C to break
Now running loop 10 |
Test passed
Testing Flash
Test passed
YAMON>
```

図8　メイン・メモリとフラッシュ・メモリのテストが正常終了したようす

```
Flash memory size =             8 MByte
SDRAM size =                    128 MByte
First free SDRAM address =      0x8008f0bc

YAMON> load asc:
About to load asc://tty1
Press Ctrl-C to break
Start dump from terminal program
Start = 0xa0100000, range = (0xa0100000,0xa01006bf), format = SREC
YAMON> go . 1
User application returned with code = 00000000
YAMON> go . 2
User application returned with code = 00000000
YAMON> go . 3
User application returned with code = 00000000
YAMON> go . 4
User application returned with code = 00000000
YAMON> go . 0
User application returned with code = 00000000
YAMON> go . 2
User application returned with code = 00000000
YAMON> go . 1
User application returned with code = 00000000
YAMON>
```

図10　モニタのload，goコマンドを実行した画面

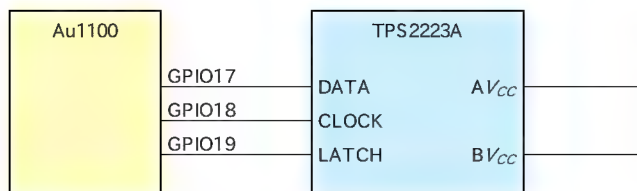


図9　Au1100とTPS2223Aの接続図

```
go . 1
```

上記のコマンドを実行したときのAVCCの波形が図11になります．AVCCが3.3Vになっていることが確認できます．

もし動作に不具合があれば，以下のように原因を見つけ対処します．

- 制御信号（DATA，CLOCK，LATCH）が正しくTPS2223Aに接続されているか，ほかの信号と短絡していないか
- 出力信号（AVCC，BVCC）がほかの信号と短絡していないか

接続に問題があったり短絡がみつかった場合にはメイン・メモリのところでも説明しましたが，パターンをカットや線材を使用して配線を直します．

この段階のハードウェアのデバッグにはソフトウェアの知識も必要です．とくに周辺機能（USBなど）を動作させるためにはプロトコル・スタックなどが必要なため，デバッグ用のプログラムを作成するのは困難です．OSを動作させて機能の確認を行うのが良いでしょう．

デバッグ事例

「E!Kit-1100」を開発するにあたり，実際にあったバグとその対処方法をいくつか紹介します．

リスト 2　動作確認用のプログラム(tps.c)

```
/*
 TPS2223A test program for E!Kit-1100
 Device Drivers Limited
 02/20/04
*/

typedef unsigned char UINT8;
typedef signed char INT8;
typedef unsigned short UINT16;
typedef signed short INT16;
typedef unsigned int UINT32;
typedef signed int INT32;
typedef unsigned long long UINT64;
typedef signed long long INT64;
typedef UINT8 bool;

#define TPS_CFDATA_SET   ((*(volatile UINT32 *)0xB1900108) = 0x20000)
#define TPS_CFDATA_CLR   ((*(volatile UINT32 *)0xB190010C) = 0x20000)
#define TPS_CFCLOCK_SET  ((*(volatile UINT32 *)0xB1900108) = 0x40000)
#define TPS_CFCLOCK_CLR  ((*(volatile UINT32 *)0xB190010C) = 0x40000)
#define TPS_CFLATCH_SET  ((*(volatile UINT32 *)0xB1900108) = 0x80000)
#define TPS_CFLATCH_CLR  ((*(volatile UINT32 *)0xB190010C) = 0x80000)

#define TPS_LENGTH 11
#define TPS_SHDN    0x100
#define TPS_AVCC0   0x000
#define TPS_AVCC3   0x004
#define TPS_AVCC5   0x008
#define TPS_BVCC0   0x000
#define TPS_BVCC3   0x080
#define TPS_BVCC5   0x040

void tps_write(UINT16 d)
{
    int i;

    TPS_CFDATA_CLR;
    TPS_CFCLOCK_CLR;
    TPS_CFLATCH_CLR;

    for(i = 0; i < TPS_LENGTH; i++)
    {
        if(d & (1 << (TPS_LENGTH - 1)))
            TPS_CFDATA_SET;
        else
            TPS_CFDATA_CLR;
        TPS_CFCLOCK_SET;
        TPS_CFCLOCK_CLR;

        d <<= 1;
    }

    TPS_CFLATCH_SET;
    TPS_CFCLOCK_SET;
    TPS_CFCLOCK_CLR;
    TPS_CFLATCH_CLR;
}

int main(int argc, char **argv)
{
    int p;

    if(argc < 2) return -1;

    p = atoi(argv[1]);

    if(p == 0)
        tps_write(TPS_SHDN);
    else if(p == 1)
        tps_write(TPS_SHDN | TPS_AVCC3);
    else if(p == 2)
        tps_write(TPS_SHDN | TPS_AVCC5);
    else if(p == 3)
        tps_write(TPS_SHDN | TPS_BVCC3);
    else if(p == 4)
        tps_write(TPS_SHDN | TPS_BVCC5);
    else
        return -1;

    return 0;
}
```

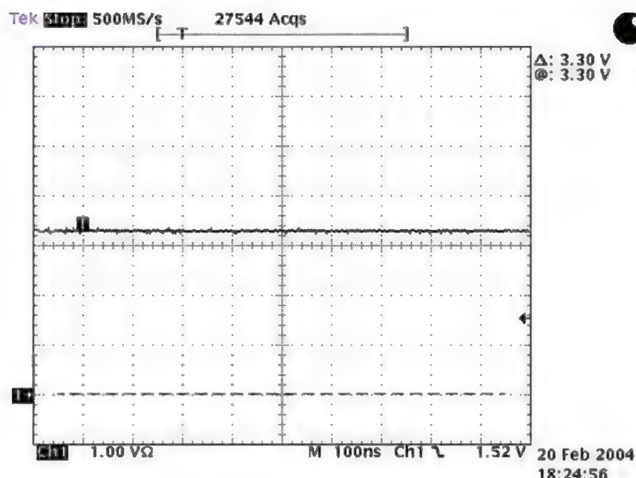


図 11　AVCC の波形



写真 3　プローブの接続

● 事例 1 ：フラッシュ・メモリの不具合

デバッグ中にフラッシュ・メモリの消去ができないことがわかりました．そこで写真 3 のようにオシロスコープのプローブを接続し，JTAGデバッガでフラッシュ・メモリにライト・アクセスしたときの波形を確認することにしました．

```
DATA.SET 0xBFC00000 %LONG 0xAA
```

上記の data.set コマンドを実行したときの波形が図 12 です(上からそれぞれ RBE0#, RWE#, RCS0#)．E!Kit-1100のフラッシュ・メモリは 16ビット・バスで Au1100に接続されているので，32ビット・アクセスについては 2回サイクルが発生するのは正常です．

```
DATA.SET 0xBFC00000 %WORD 0xAA
```

COLUMNS

モニタ・プログラム

モニタ・プログラムは一般的にCPUや半導体，ボード開発メーカなどが有償/無償で提供している，CPUの動作確認やまたデバッグするのに必要な機能が入っているプログラムです．

モニタ・プログラムのおもな特徴としては，次のようなものがあります．

1) 動作対象のCPUの機能をサポートし，動作することが(ある程度)保障されている
2) ほかにOSや設定ツールがなくても，それ自体単体で動作できる
3) ROMやRAMに配置して動作できる小さなサイズ
4) シリアル・ポートやネットワークなどからほかのプログラム(オブジェクト)モジュールをロードして実行する機能をもつ
5) メモリの表示，書き換えといった簡単なデバッグ機能をもっている

モニタ・プログラムはバイナリだけでメーカなどから提供される場合から，ソースコードやコンパイラ，開発環境やデバッガまで含めて統合環境として提供される場合までさまざまです．またすでに製品開発を経験している企業では，自社製のモニタ・プログラムを使用したり，開発に先立って自前でモニタ・プログラムを開発したり，個別に調達する場合もあります．E!Kit-1100の開発では，MIPSアーキテクチャCPUの標準的なモニタとして，ネットワーク上でも無償配布されているYAMONを使用しています．モニタのおもなコマンドを表Bに示します．

表B　モニタのおもなコマンド

コマンド	動　作
dump	指定したアドレスを表示
edit	指定したアドレスを編集
load	LAN，シリアル・ポート，CF※からプログラムをロード
go	メモリ上のプログラムを実行
test	メイン・メモリとフラッシュ・メモリのテスト

※CFからのプログラムのロードは当社が独自に追加した機能です．

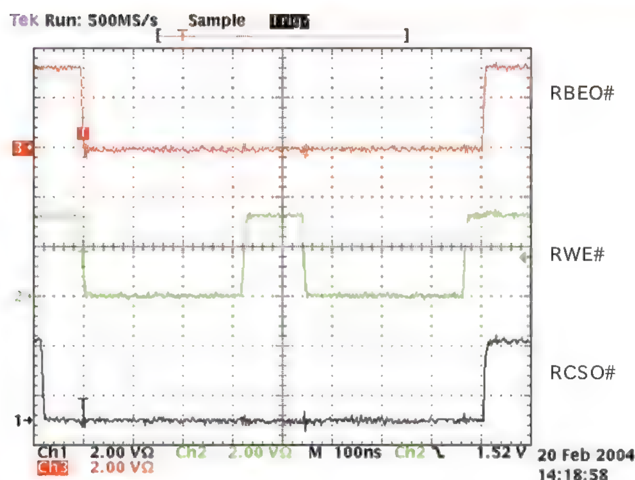


図12　data.setコマンドを実行したときの波形(1)

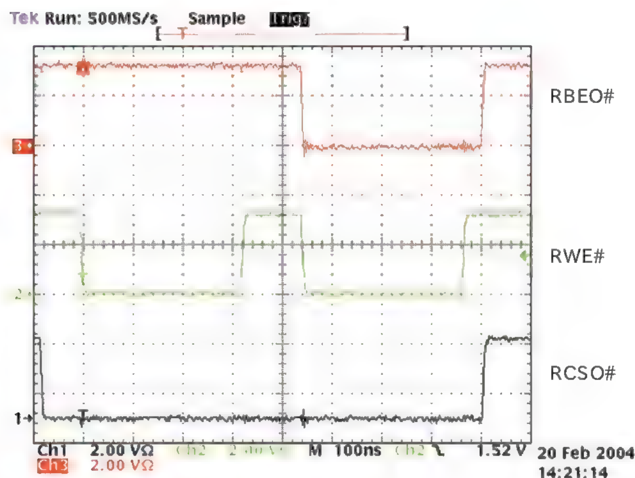


図13　data.setコマンドを実行したときの波形(2)

上記の`data.set`コマンドを実行したときの波形が図13です．これでもアクセス・サイクルが2回発生しています．これではフラッシュ・メモリが正常に動作しません．

この現象は，Au1100はMIPSアーキテクチャなので16ビット・アクセスも内部バス上では32ビット・アクセスになり，外部の16ビット・バスではAu1100の機能で16ビット・アクセス2回になってしまうのでは？と推測し，図14のような回路変更で対処することを考えました．この回路はRBE0#＝"L"のときだけRWE#が出力(＝"L")されるようにしたものです．

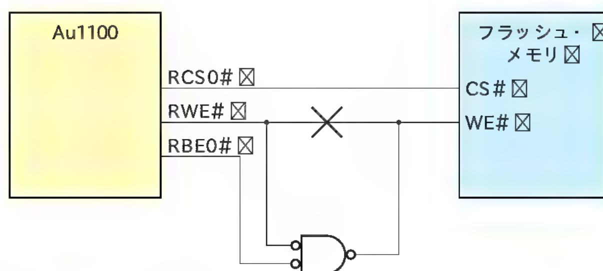


図14　対処した回路案

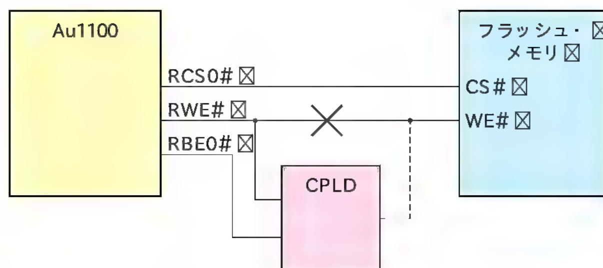


図15　対処した回路

この対処を行うには部品(2入力ORゲート)を追加する必要があります．しかし，すでに実装済みのCPLDにRWE#，RBE0#信号が接続されていたので，CPLDの空き端子を使用して対処しました．その回路が図15です．対処方法としてはフラッシュ・メモリのWE#ピンを持ち上げて信号線から切り離し(配線が内層を通っていてカットができなかったため)，点線部分の配線を追加しました．そしてCPLDの内部回路を変更します．CPLDなどのような内部回路を変更可能な部品はたいへん便利です．

しかし，その後Au1100のデータ・シートをよく確認するとAu1100の設定を変更することで回避できることがわかりました．

```
DATA.SET 0xBFC00000 %WORD 0xAA
```

初期設定値を変更した後，上記のdata.setコマンドを実行したときの波形が図16です．ライト・アクセスが1回になっていることが確認できます．

この対処でフラッシュ・メモリも無事動作するようになりました．もちろんモニタの初期化コードの修正も忘れずに行います．

このように，何か問題があった場合，自分が想定していること(今回の場合は16ビット・アクセス時のAu1100の動作)がまちがっている場合が多々あります．したがって，思い込みでデバッグせず，測定器などですぐに確認することが重要です．

● 事例2：電源が一瞬だけONになる

「E!Kit-1100」に電源を接続したとき，一瞬だけ電源回路がONになることがわかりました．使用上とくに問題ありませんが，自分が想定していない動作をしている場合には，その原因を調査することが必要です．

図17が試作機の電源制御回路です．

- 「E!Kit-1100」に電源を接続(DC5Vが供給)されるとDフリッ

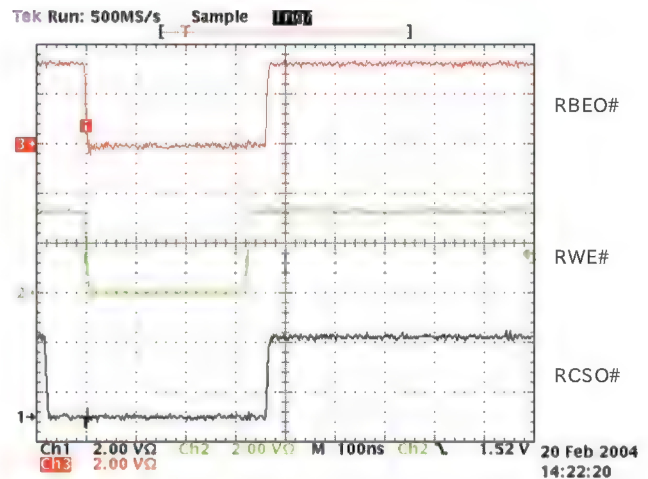


図16　data.setコマンドを実行したときの波形(3)

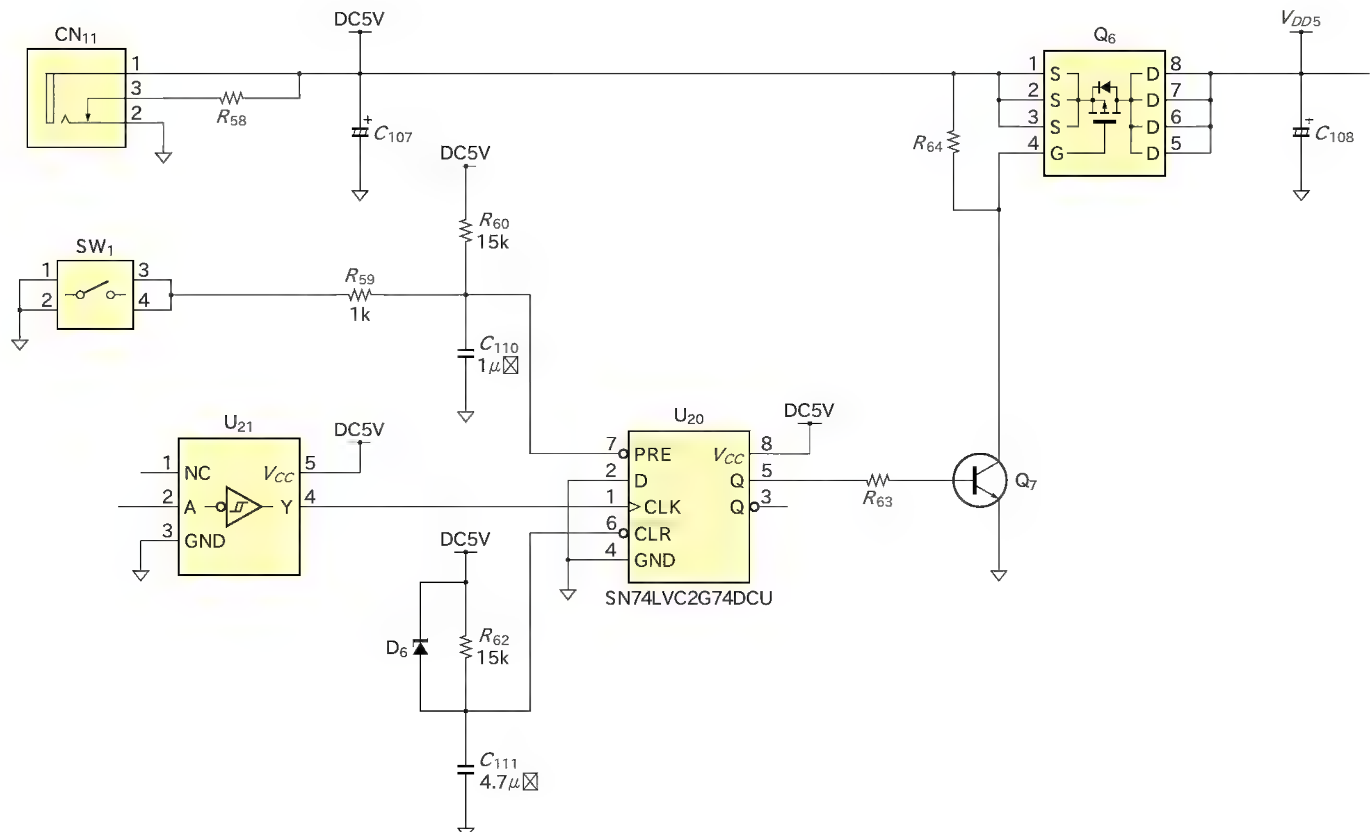


図17　試作機の電源制御回路

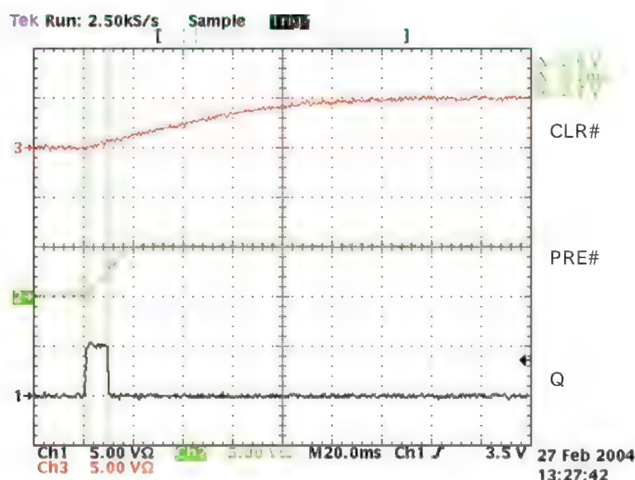


図 18　対策前の波形

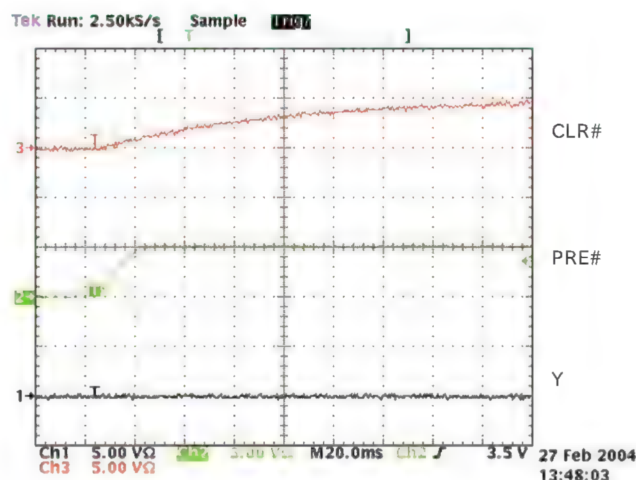


図 20　対策後の波形

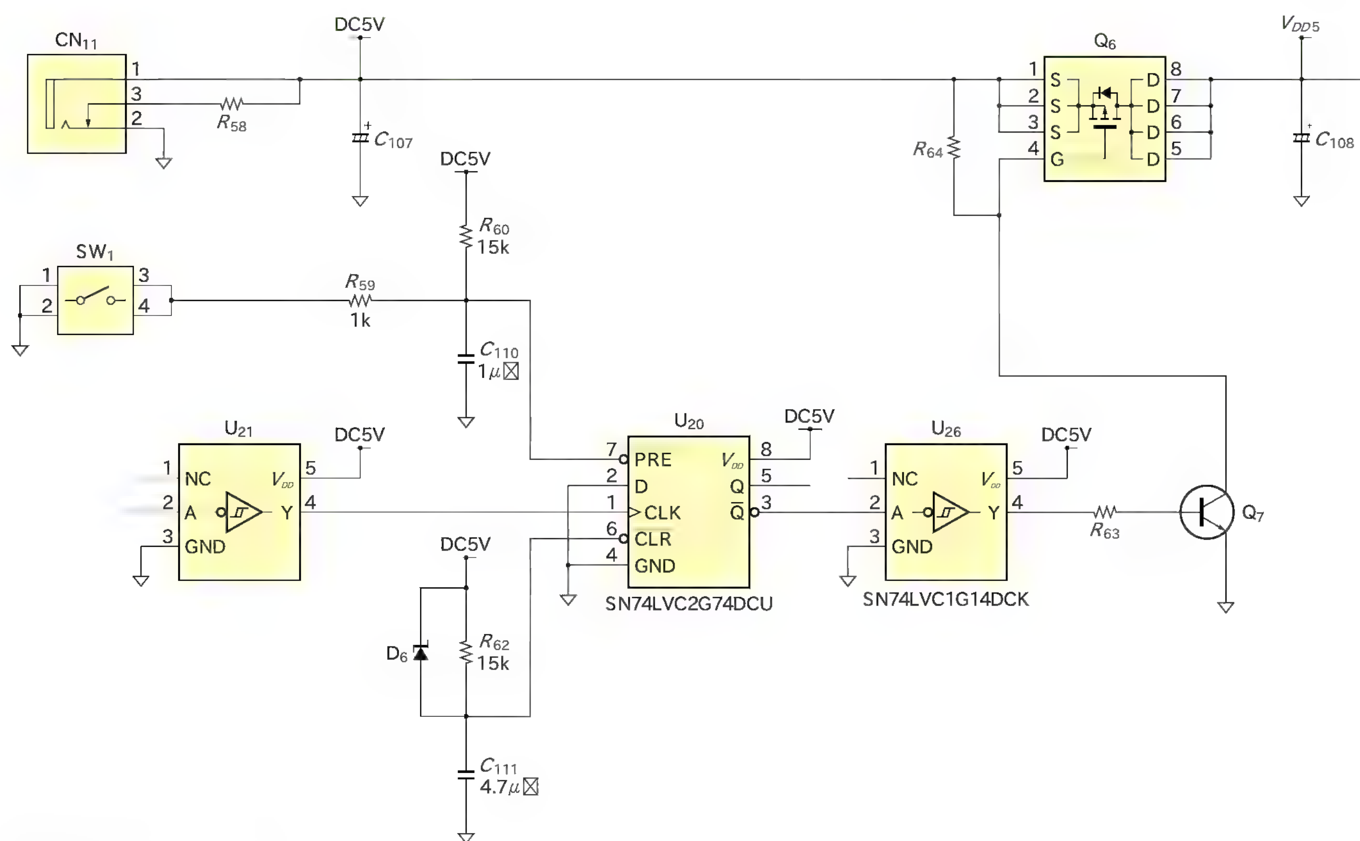


図 19　対策後の電源制御回路

プフロップ(U_{20})の CLR# が" L", Q が" L"になり電源 V_{DD5}) は必ず OFF になる

- 電源スイッチ(SW_1)が押されるとフリップフロップ(U_{20})の PRE# が" L", Q が" H"になり電源(V_{DD5})が ON になる
- ほかの回路から U_{21} を経由して D フリップフロップ(U_{20})の CLK に立ち上がりエッジの信号が入力されると電源(V_{DD5})が OFF になる

しかし，電源スイッチ(SW_1)と D フリップフロップ(U_{20})の間にあるチャタリング防止用の抵抗，コンデンサによって D フリップフロップ(U_{20})の PRE# が" H"になるのが遅れ，PRE# と CLR# がともに" L"になる期間があり，そのとき Q，Q# がともに" H"になることがわかりました．そのときの波形が図 18 です(上から U_{20} の CLR#，PRE#，Q)．D フリップフロップ(U_{20})の仕様については表 2 を参照してください．

表2 Dフリップフロップ(U20)の仕様

INPUT				OUTPUT	
PRE#	CLR#	CLK	D	Q	Q#
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H	H
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	前の状態を保持	

そこでDフリップフロップ(U20)のQではなく，Q#を反転して使用することにしました．図19が対策後の電源制御回路です．そのときの波形が図20です(上からU20のCLR#，PRE#，U26のY)．

この不具合は，使用するうえでとくに問題はないと判断し，改造費用などを考慮して試作機では未対応とし，製品の回路にのみに対応しました．このような判断もハードウェア開発には必要です．

まとめ

ハードウェアのデバッグについて一通り説明しました．ハードウェアのデバッグが必要になる時期として「ハードウェア開発時」，「ソフトウェア開発時にハードウェアの不具合が発覚したとき」，「製品出荷後にハードウェアの不具合が発覚したとき」などが考えられます．時期が後のほうになるほど影響が大きくなり，とくに製品出荷後にハードウェアの不具合があった場合は，回収費用なども含めて大きな損害になります．製品出荷前の入念なデバッグがたいへん重要です．

最後になりますが，組み込みシステムでのハードウェアのデバッグがどういうことをするのか，この記事が理解するきっかけになれば幸いです．

COLUMN

CPLD

CPLD(Complex Programmable Logic Device)とは，完全にプログラマブルなAND/ORアレイとマクロセルを組み合わせたものです．AND/ORアレイは多様なロジック機能を，マクロセルは機能的なブロックです．これらを自在に組み合わせることによって組み合わせロジックやシーケンシャル・ロジックが実現可能なデバイスです．

基板実装後でも回路が変更可能な，不揮発性デバイスであることが一般的なCPLDの特徴です．「E!Kit-1100」でもCFやUSBの制御回路にCPLDを採用しています．回路変更も容易にできますし，空きピンなどを使用すればデバッグによる回路の追加なども容易になりたいへん便利です．

参考文献
(1) *AMD Alchemy Solutions Au1100 Processor Data Book/30362b*, AMD.
(2) *TPS2220A, TPS2223A, TPS2224A, TPS2226A CARDBUS POWER-INTERFACE SWITCHES FOR SERIAL PCMCIA CONTROLLERS*, Texas Instruments.
(3) *SN74LVC2G74 SINGLE POSITIVE-EDGE-TRIGGERED D-TYPE FLIP-FLOP WITH CLEAR AND PRESET*, Texas Instruments.

■ E!Kit-1100の入手先
(株)デバイスドライバーズ E-KIT事業部
TEL ：042-363-8294
FAX ：042-363-8255
URL ：http://e-kit.jp/
E-mail ：e-kit@devdrv.co.jp

かわもと・やすひさ (株)デバイスドライバーズ

4. 異なった環境で動作するソフトウェアをPCで開発する

組み込み向けクロス開発環境の構築

中村 憲一

組み込みシステムでは，低消費電力を実現するためにCPUを低速で動作させ，極小のメモリ容量で動作させることがある．そのようなシステムを開発するために，組み込みシステム自体でコンパイラを動作させ，開発を行う「セルフ開発」は難しいといえる．

そこでPC上で開発を行い，作成したプログラムを組み込みシステム上へ送信して動作させる「クロス開発」が行われる．x86でWindowsを採用したPC上で，組み込みシステムで多く使われるARM，SH，MIPS，PowerPC…用のコードを作成するのである．

このような，組み込みシステム独特の世界，「クロス開発」について取り上げる．　（編集部）

さて，読者の皆さんはWindows，UNIX，Linuxなどの OS上で動くアプリケーション・プログラムを作成したことがありますか？　最近では，高校や大学の授業で実際にPCを使用してある OS上で動作するアプリケーション・プログラムのプログラミングを習った方も多いでしょう．ひと言でプログラミングといっても，開発言語には数え切れないくらいの種類が存在しますが，多くの皆さんはC言語について学習されたかと思います．また中には，C++言語やJava言語などのオブジェクト指向言語を学習された方もいることでしょう．

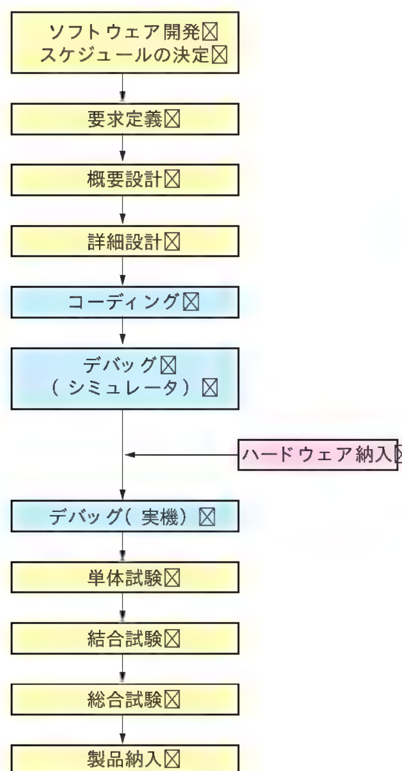


図1
組み込みソフトウェア開発の例

最近では，携帯電話，カーナビ，HDD/DVDレコーダ，ディジタル・カメラなどのような組み込みシステムにもOSが載っており，開発者はアプリケーション・プログラムのみを開発すればよいというような組み込みシステムも多くなってきています．しかし，このような高級なシステムは前もってOSが正しく動作していることが前提となっており，OSが使われないシステムもまだまだたくさん存在します．とくに，8ビットや16ビットのプロセッサを採用し，ROMやRAMの容量が限られているようなシステムでは，OSが使われることがほとんどないといっても過言ではないでしょう．具体的には，炊飯ジャー，電気ポット，電子レンジ，全自動洗濯機など，電圧・電流・時間・温度・液量・気圧・モータなどを制御する比較的単純なシステムです．中でも自動車などは，このような小さなシステムが集まった巨大な組み込みシステムといえるでしょう．

本章では，このようなOSの存在しない組み込みシステムにおけるソフトウェア開発の基礎について説明します．

PCで開発できる組み込みシステム

組み込みシステムを開発するとなると難しそうに思えるでしょうが，じつは読者の皆さんが持っているほとんどのPCで開発が可能であり，実際に，ほとんどの開発現場で市販のPCがそのまま利用されています．

では，どのようにして組み込みソフトウェアの開発が進むのでしょうか？　ある組み込みソフトウェア開発の例を図1に示します（ちなみに，このような開発モデルは，上流から下流へ水が流れ落ちるようすに似ていることからウォータ・フォール・モデルと呼ばれている）．ここでは，コーディングとデバッグ行程について着目しましょう．デバッグ工程がシミュレータ上と実機上の2工程存在するのが組み込みソフトウェア開発の特徴です．組み込みシステムの開発では，ソフトウェアとハードウェアの開発が平行して行われるため，このような開発工程

になることが多いのです．また，この図にはありませんが，じつは，ほとんどの開発プロジェクトにおいてコーディング工程に入るまでに設計工程と並行して，

● 開発ツールの調査，評価，選定，導入作業
● 開発担当者向けの開発ツールのトレーニング

が行われているのです．組み込みシステムにおける開発ツールとはいったいどのようなツールなのでしょうか?

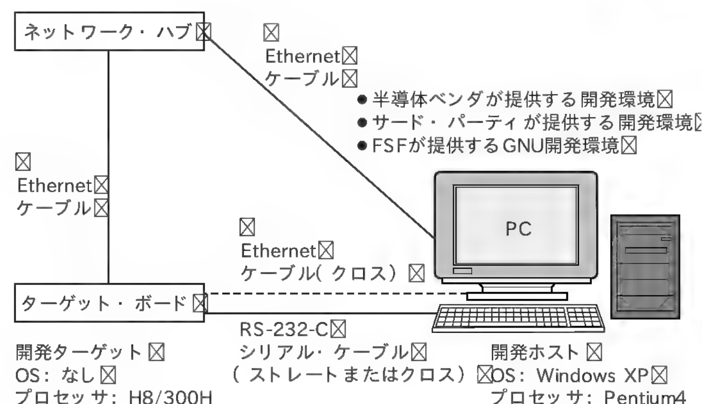


図2　クロス開発の概念図

クロス開発とは?

先ほど市販されているPCで開発ができると書きましたが，開発ホストとなる市販のPCに採用されているプロセッサとオペレーティング・システムの組み合わせは，ほとんどがIA-32アーキテクチャのプロセッサ(米Intel社のPentium4など)と米Microsoft社のWindows(2000/XPなど)です．

一方，開発ターゲットとなる組み込みシステムに採用されているプロセッサは，H8，SH，ARM，PowerPC，MIPSなど多種多様です．本当にこのようなPCで，異なるアーキテクチャのプロセッサ上で動作するプログラムを開発できるのでしょうか?

そこで，このようなプロセッサもOSも異なるホストとターゲット間での開発を可能にするのがクロス開発と呼ばれる開発手法であり，それを実現するのがクロス開発ツールなのです(図2)．クロス開発ツールには，

● 半導体ベンダが提供する開発環境(表1)
● サード・パーティが提供する開発環境(表2)
● Free Software Foundation(FSF)が提供するGNU開発環境

などがあります．

したがって，H8やSHを搭載したターゲットであれば，半導体ベンダが提供するHEW(High-performance Embedded Workshop)などの開発環境を使用すればよいことがわかります．しかし，半導体ベンダが提供する開発環境を使用した場合，ほかの半導体ベンダのプロセッサを採用することになったときは，また新たに開発環境を導入しなければなりません．さらに使い勝手も異なるので，開発者のトレーニングも必要になってしまいます．

そこで登場するのがサード・パーティが提供する開発環境です．サード・パーティ製の開発環境であれば，たいていの場合，複数のターゲットに対応しており，異なるターゲットにおいても操作方法に変化がないため，プロセッサの変更にも柔軟に対応できます．

しかし，サード・パーティ製の開発環境も良いことばかりではありません．サード・パーティ製の開発環境は優れている反面，一般的に高価なので，製品開発コストの低下を強いられているユーザにとってはその販売価格が問題になってきます．プログラム・サイズの増大とともに開発者の人数も増え，開発者の人数分だけライセンスを購入することが非常に難しくなっているのです．よって，最近では，開発環境を販売せずにリースを行うサード・パーティも現れてきています．

COLUMN

用語解説

本章で登場した組み込み関連用語について，解説します．

● クロス開発…PCとターゲット・ボードのように異なるプラットホーム間で開発を行うこと
● FSF…リチャード・ストールマン氏が率いるフリーソフトウェア財団
● GNU…GNU is Not UNIXの略．FSFが推進するGNUプロジェクト
● GNUソフトウェア…GNUプロジェクトにより開発されたソフトウェア
● binutils…バイナリ・ユーティリティ(as，ld，nm，objdump，objcopyなどのコマンドを含む)
● GCC…GNUコンパイラ・コレクション(gcc，g++などのC，C++コンパイラを含む)
● GDB…GNUデバッガ
● GDBスタブ…PC上で動作するデバッガGDBの通信相手としてターゲット・ボード上で動作する小さなプログラムのこと．おもにデバッグの際に使用する
● Newlib…組み込みシステム向けCライブラリ
● 統合開発環境(IDE:Integrated Development Environment)エディタ，コンパイラ，アセンブラ，デバッガなどを統合した開発環境のこと
● パワーオン・リセット…電源投入時に実行されるリセットのこと
● ROM化…ROMに書き込む(焼き込む)こと

表1　半導体ベンダ製の開発環境の例

プロセッサ	開発環境	開発元
H8，SH	High-performance Embedded Workshop（HEW）	（株）ルネサステクノロジ
M32R	M3T-CC32R	
V850	SP850	NECエレクトロニクス（株）
ARM	RealView Developer Suite	英ARM社
TX19	東芝統合開発環境（IDE）	東芝（株）
XStormy16	三洋マイコン開発ツール	三洋電機（株）
AM1，AM2，AM3	PanaX Series	松下電器産業（株）
FR，FR-V	SOFTUNE	富士通（株）

表2　サード・パーティ製の開発環境の例

開発環境	開発元	備　考
MULTI2000	米Green Hills Software社	
CodeWarrior	米Metrowerks Corporation	
eBinder	イーソル（株）	
OPENplus	ガイオ・テクノロジー（株）	
exeGCC	京都マイクロコンピュータ（株）	GNUツール
GNUPro	米Red Hat社	GNUツール
GNUWing	米Upwind Technology社	GNUツール
KPIT GNU Tools	印KPIT Cummins Infosystems社	GNUツール

表3　GCCが対応しているプロセッサの例

アーキテクチャ	プロセッサ
H8	H8/300L，H8/300，H8/300H Tiny，H8/300H，H8S
SuperH	SH-1，SH-2，SH-2e，SH-3，SH-3e，SH-4
ARM	ARM7，ARM9，ARM9E，ARM10E，StrongARM，Xscale
MIPS32,64	R3000，R4000，V_R4100，V_R4300，V_R5000
	TX19，TX39，TX49
PowerPC	PowerPC 405，PowerPC 440，MPC74xx，MPC7xx，MPC60x，MPC824x
	PowerQUICC，PowerQUICC II，PowerQUICC III
IA-32	80286，80386，80486，K5，K6
	Pentium，PentiumPro，Pentium II，Pentium III，Pentium4，Athlon
IA-64	Itenium
その他	M32R，D10V，D30V，AM2，AM3，FR，FR-V
	XStormy16，V850，M68000，Alpha，SPARCなど

表4　GCCが対応しているオペレーティング・システムの例

アーキテクチャ	オペレーティング・システム
Alpha	Linux，NetBSD
SPARC	Sun OS，Solaris，Linux，NetBSD
IA-32	Windows 95/98/NT/2000/XP
	Linux，Solaris，FreeBSD，OpenBSD，NetBSD
IA-64	Linux
PowerPC	Linux，AIX，NetBSD
MIPS	IRIX，Linux，NetBSD
ARM	Linux，NetBSD
SuperH	Linux，NetBSD

GNU開発環境とは？

半導体ベンダやサード・パーティが提供する開発環境が抱える問題を解決するのが，米FSFが運営するGNUプロジェクトにて提供されるGNU開発環境です．GNU開発環境は無償で配布されているため，個人で学習する際にも非常に適しています．また，表2に示したサード・パーティからも，有償サポートなどの付加価値を付けて提供されているので，商用での使用においてもまったく問題ないでしょう．開発ホストやターゲットが異なっていても同じ操作体系で，ライセンスを購入することなく使用できるのがGNU開発環境なのです．

GNU開発環境には，binutils（バイナリ・ユーティリティ），GCC（GNUコンパイラ・コレクション），GDB（GNUデバッガ）などが含まれます．GCCが対応しているプロセッサの例を表3に，GCCが対応しているオペレーティング・システムの例を表4に示します．ホストとターゲットの組み合わせがメジャーなものであれば，おそらくほとんどの開発に対応できるでしょう．

GNUプロジェクトとは？

1984年にリチャード・ストールマン氏により開始されたプロジェクトであり，1985年に同氏により設立されたフリー・ソフトウェア開発のための非営利団体です．フリー・ソフトウェア財団（FSF）により運営されています．GNUプロジェクトでは，プロジェクト開始以来，UNIXに似たフリー・ソフトウェアの完全なオペレーティング・システム，GNUシステムを開発してきました（GNUとは「GNU's Not UNIX（GNUはUNIXではない）」の再帰頭字語であり，「グニュー」と発音される）．現在，組み込みシステムにも多く採用されているLinuxシステムも，じつはほとんどがGNUソフトウェアで構成されています．よって，正確にはGNU/Linuxシステムと呼ばれるべきものなのです．GNUプロジェクトの詳細は，Webサイト（`http://www.fsf.org/`）を参照してください（図3）．

● binutilsとは？

binutilsとは，バイナリ・ユーティリティのことで，

- `as`：アセンブラ
- `ld`：リンカ
- `nm`：オブジェクト・ファイル中のシンボルを表示する
- `objdump`：オブジェクト・ファイルの内容の詳細を表示する
- `objcopy`：オブジェクト・ファイルをSレコード形式など，ほかのフォーマットに変換する
- `strip`：オブジェクト・ファイルからシンボル情報を削除する

などのコマンドやリンカ・スクリプト（リンク・アドレス，セクションなどを指定する）が含まれています．基本的な操作は

コマンド・ラインで行いますが，asやldなどについては通常コンパイラから自動的に呼び出されます．

● GCCとは？

GCCとは，GNUコンパイラ・コレクションのことで，GNU Cコンパイラ(gcc)，GNU C++コンパイラ(g++)などを含んでいます．基本的な操作はコマンド・ラインで行いますが，統合開発環境(IDE：Integrated Development Environment)と併用することにより，コマンド・ラインを意識せずに利用することができます．

● GDBとは？

GDBとは，GNUデバッガのことで，コマンド・ラインのデバッガです．基本的な操作はコマンド・ラインで行いますが，コマンド・ラインでの操作が苦手な方には，InsightやDDDなどのようなGUIフロント・エンドを利用することもできます．

● Newlibとは？

C言語の教科書などで有名な"Hello World"プログラムでは，printf()関数を使用して標準出力に文字列を出力しています．一方，組み込みシステムには，標準のprintf()関数や標準出力というものは存在しません(組み込みシステムではシリアル・ポートに出力することが多いので，シリアル・ポートが標準出力であるといっても過言ではない)．よって，組み込みシステムにおいては，関数は自分で用意しなければなりません．

では，それらを毎回作成しなければいけないのでしょうか？先ほど組み込みシステムには標準のprintf()関数などは存在しないと書きましたが，同じ機能をもつ関数を毎回製品ごとに自作することはそもそも不可能です．そこで，あらかじめ作成したり，社外から調達することになるのですが，幸いにもこれらの関数群を収録したフリーのライブラリが世界中にたくさん存在します．ここでは，その中の一つであるNewlibを紹介します．

Newlibは，米Red Hat社が運営しているWebサイト(http://sources.redhat.com/newlib/)で公開・配布されている組み込みシステム向けの小さなライブラリで，現在は，米Red Hat社のJeff Johnston氏とTom Fitzsimmons氏がメンテナンスを行っています．

NewlibのライセンスColor・ファイル(COPYING.NEWLIB)を読むと，米カリフォルニア大学バークレー校，米AT&T社，米Advanced Micro Devices社，米Red Hat社，米Sun Microsystems社，米Hewlett-Packard社，米SuperH社，米Intel社，FreeBSDプロジェクト，米Free Software Foundationなどが著作権を所有しており，各社・各団体がその利用を許可していることがわかります．

このようにNewlibは各社・各団体が共同で開発したライブラリであり，ランタイム・ライセンスが不要であるという特徴をもっています．よって，自社で組み込み機器向けの小さなライブラリを所有していない場合は，必須のライブラリともいえるでしょう．

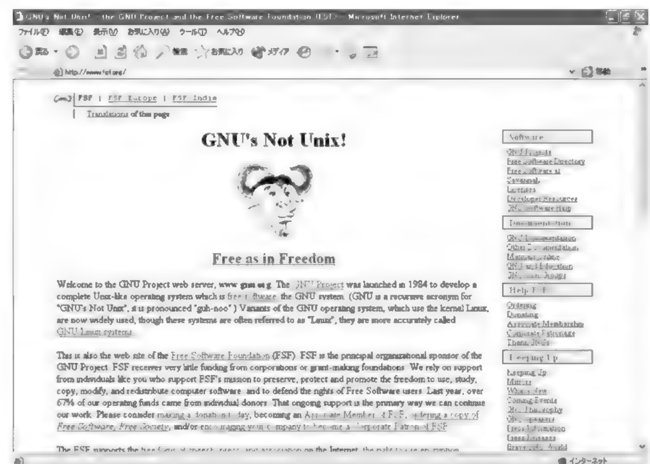


図3　米Free Software FoundationのWebサイト(http://www.fsf.org/)

GNU開発ツールのビルド

それでは，米FSFのWebサイトからGNU開発環境を入手し使用してみることにしましょう．どうでしょうか？　必要なバイナリ・コードが見あたらないでしょうか？　そうです．じつは，米FSFのWebサイトからはクロス開発に使用するためのバイナリ・コードは配布されていません．というのも，開発するホスト，そしてターゲットとなるプロセッサの組み合わせは，無数に存在します．よって，これらのすべての組み合わせについてバイナリ・コードを配布することは不可能なのです．しかし，ここであきらめないでください．バイナリ・コードがなくてもソース・コードが配布されているので，ソース・コードから簡単にバイナリ・コードをビルド(構築)することができるのです．

ここでは，多数の読者が利用していると思われるWindows 2000/XP環境を例にビルドし，開発環境を構築してみます．

開発環境の構築手順は，以下のとおりです．

- Cygwin環境のダウンロードとインストール
- binutils, GCC, Newlib, GDBのダウンロードとコンフィグレーション，ビルド，インストール
- GDBスタブのROMへの書き込み

● Cygwin環境のダウンロードとインストール

Cygwin環境とは，Windows上でUNIXに似た環境を提供する環境であり，cygwin1.dllというダイナミック・リンク・ライブラリ(DLL)を中心として，シェルやエディタなどさまざまなツールが提供されています．2004年3月1日現在，CygwinプロジェクトのWebサイト(図4，http://cygwin.com/)からバージョン1.5.7-1のCygwin DLLが提供されています．初めてのダウンロードとインストールの際は，setup.exeをダウンロードして実行し，指示に従ってダウンロードとインストー

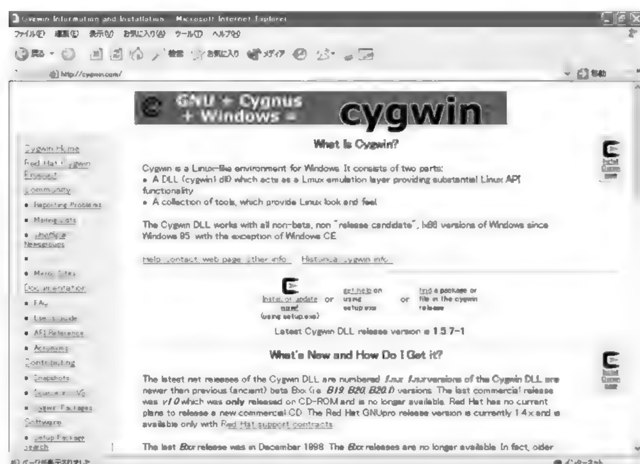


図4　Cygwinプロジェクトの Webサイト(http://cygwin.com/)



図5　Cygwinのシェル

```
$ mkdir -p /usr/local/src/gnu
```

図6　mkdirコマンドを使用したディレクトリの作成

```
$ cd /usr/local/src/gnu
$ tar jxvf binutils-2.14.tar.bz2
$ tar jxvf gcc-3.3.3.tar.bz2
$ tar zxvf newlib-1.11.0.tar.gz
$ tar jxvf insight-6.0.tar.bz2
$ cp -r newlib-1.11.0/newlib ./gcc-3.3.3/
$ cp -r newlib-1.11.0/libgloss ./gcc-3.3.3/
```

図7　ファイルの解凍・展開

ルを行います．無事インストールが終わると，(インストールの終了時に選択した場合)デスクトップ上にCygwinというショート・カットができるので，ダブルクリックしてシェルを起動してください．lsなどのコマンドが使用できるように環境が整っているはずです(図5)．GUIを用いた統合開発環境に慣れている方にとっては，いきなりシェルがでてきて驚くかもしれませんが，ここでは基礎について解説しているので，シェルでの操作を前提とします．もちろん，GNU開発環境にもGUIを用いた統合開発環境がたくさん用意されているので，プロジェクトの開発スタイルにあった環境を構築すると良いでしょう．

● binutils，gcc，newlib，gdbのダウンロードとコンフィグレーション，ビルド，インストール

▶ ソース・コードのダウンロードと解凍・展開

まず，mkdirコマンド(図6)で/usr/local/src/gnuディレクトリを作成し，その下に，

- binutils-2.14.tar.bz2
- gcc-3.3.3.tar.bz2
- newlib-1.11.0.tar.gz
- insight-6.0.tar.bz2

を各配布元またはミラー先からダウンロードして保存します．各パッケージの配布元を表5に示します．

表5　各パッケージの配布元

パッケージ	バージョン	配布元
binutils	2.14	http://sources.redhat.com/binutils/
GCC	3.3.3	http://gcc.gnu.org/
Newlib	1.11.0	http://sources.redhat.com/newlib/

次に，ダウンロードしたファイルを解凍・展開します(図7)．

▶ binutilsのコンフィグレーション，ビルド，インストール

GNUソフトウェアをソース・コードからビルドするためには，ほとんどの場合でコンフィグレーション(configure)，ビルド(make all)，インストール(make install)という3段階の手順が必要となります．これは，異なるホストやターゲットに対応するために実装されている便利なしくみ(Autoconfという機能)です．

では，早速，binutilsのコンフィグレーション，ビルド，インストールを行ってみましょう．今回は，ターゲットがH8/300H(オブジェクト・ファイル形式はh8300-elf)である例について説明します．そのほかのターゲットの場合は，表6に示すターゲット名を使用してください．また，インストールディレクトリは，/usr/local/gnuと仮定します．binutilsのコンフィグレーション，ビルド，インストールの手順を図8に示します．

▶ gcc, newlibのコンフィグレーション，ビルド，インストール

binutilsのインストールが完了したら，gccとnewlibのコンフィグレーション，ビルド，インストールを行います(図9)．newlibのビルドにけっこう時間がかかります．高速なプロセッサを搭載したPCでも1時間以上かかるので注意してください．

▶ gdb(Insight)のコンフィグレーション，ビルド，インストール

ここでは，Insightの例を示します(図10)．

● GDBスタブのROMへの書き込み

GDBスタブのコンフィグレーションでは，ソースコードを変

表6　アーキテクチャと対応するターゲット名の例

アーキテクチャ	ターゲット名
ARM（StrongARM，Xscaleを含む）	arm-elf
Intel x86	i386-elf
MIPS32	mipsisa32-elf
PowerPC	powerpc-eabi
SH	sh-elf
H8/300H	h8300-elf
M32R	m32r-elf
V850	v850-elf
FR	fr30-elf
FR-V	frv-elf

更することによるメモリ・マップやセクションの設定など，クロス開発についてすでに理解していることが前提となるので，ここでは，すでにビルドされたバイナリ・イメージを入手し，ROMに書き込むことにします．

H8/300用のGDBスタブは，「eCos/RedBoot for H8/300プロジェクト」のWebサイト（http://sourceforge.jp/projects/ecos-h8/）でバイナリ・イメージが配布されているので，これを利用することにします（図11）．リリース・ファイルのページからGDBスタブのROMイメージ（gdb_modules.mot.gz）をダウンロードします．あるいは，高機能なモニタであるRedBootのイメージ（redboot.mot.gz）をダウンロードしても良いでしょう．

このイメージは，（株）秋月電子通商が販売しているAKI-H8/3069Fフラッシュ・マイコンLANボード（写真1）で動作するようにビルドされています．このボードは，2004年3月1日現在，（株）秋月電子通商のWebサイト（図12，http://akizukidenshi.com/）にて8,700円という価格で通信販売が行われているので，秋葉原から遠方の企業，個人でも手軽に入手することが可能です．上記でH8アーキテクチャ用のGNU開



図11　「eCos/RedBoot for H8/300プロジェクト」のWebサイト

```
$ cd
$ mkdir build-binutils-h8300-elf
$ cd build-binutils-h8300-elf
$ /usr/local/src/gnu/binutils-2.14/configure
--prefix=/usr/local/gnu --target=h8300-elf
        --disable-nls -v 2>&1 | tee configure.out
$ make all 2>&1 | tee make_all.out
$ make install 2>&1 | tee make_install.out
$ export PATH=/usr/local/gnu/bin:$PATH
$ h8300-elf-as -v
```

図8　binutilsのコンフィグレーション，ビルド，インストール

```
$ cd
$ mkdir build-gcc-h8300-elf
$ cd build-gcc-h8300-elf
$ /usr/local/gnu/gcc-3.3.3/configure
--prefix=/usr/local/gnu --target=h8300-elf
--disable-nls --enable-languages=c,c++ --with-gnu-as
     --with-gnu-ld --with-newlib --with-gxx-include-
dir=/usr/local/gnu/h8300-elf/include -v 2>&1 | tee
configure.out
$ make all 2>&1 | tee make_all.out
$ make install 2>&1 | tee make_install.out
$ h8300-elf-gcc -v
```

図9　gccとnewlibのコンフィグレーション，ビルド，インストール

```
$ cd
$ mkdir build-gdb-h8300-elf
$ cd build-gdb-h8300-elf
$ /usr/local/src/gnu/insight-6.0/configure
--prefix=/usr/local/gnu --target=h8300-elf -v 2>&1 |
tee configure.out
$ make all 2>&1 | tee make_all.out
$ make install 2>&1 | tee make_install.out
$ h8300-elf-gdb -v
```

図10　gdb（Insight）のコンフィグレーション，ビルド，インストール

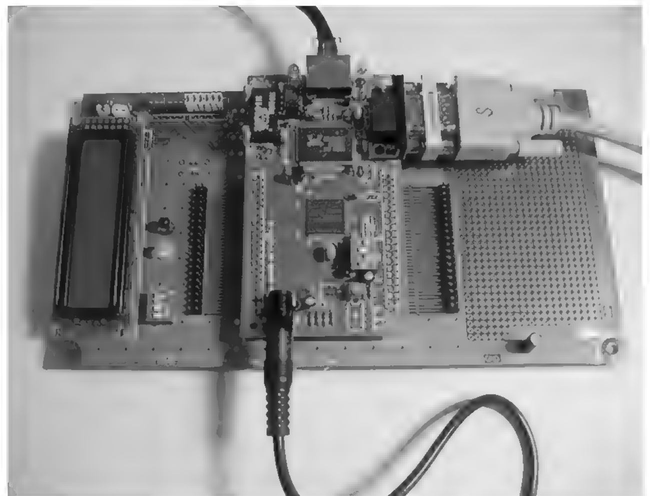


写真1　AKI-H8/3069Fボード



図12 (株)秋月電子通商のWebサイト

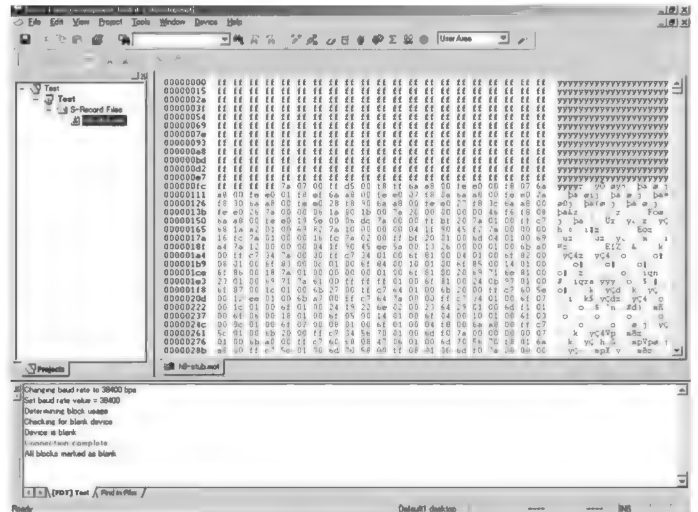


図13 ルネサスのフラッシュ開発ツールキット(FDT)

リスト1 LEDの点滅プログラム

```
#define P4DDR   (*(volatile unsigned char *)0xfee003)
#define P4DR    (*(volatile unsigned char *)0xffffd3)

int main(void){
        int i;
        P4DDR = 0xc0;           // Set bit 6 and 7 of
                                //  port No.4 for Output
        while(1){
                P4DR = 0x80;    // Write bit 7 to turn on
                                //  Green LED
                i=1000000;
                while(i>0) i--; // Wait for a moment
                P4DR = 0x40;    // Write bit 6 to turn on
                                //  Red LED
                i=1000000;
                while(i>0) i--; // Wait for a moment
        }
}
```

発環境を構築したので，実ターゲット・ボードとしてこのボードを使用することにします．私見ですが，H8アーキテクチャは組み込みシステムの初心者にとっては理解しやすいアーキテクチャであり，AKI-H8/3069Fフラッシュ・マイコンLANボードはLED，DIPスイッチ，LCDが実装されたマザーボードも付いているため，いろいろな実験を行いやすいボードだと思います．しかし，組み立てる際にはんだ付け作業を行う必要があるので，はんだ付け作業が苦手な方は，ハードウェアを得意とする同僚や知人に依頼しましょう．おそらく喜んでやってくれることでしょう．

ターゲット・ボードへGDBスタブあるいはRedBootのROMイメージを転送し，フラッシュ・メモリに書き込みます．ターゲット・ボード上のフラッシュ・メモリへの書き込みには，個人で使用する場合は，ボードとともに提供されている`h8write`というコマンドを使用することができます．また，商用で使用する場合は，ルネサスのフラッシュ開発ツールキット(FDT)を利用することができます(図13)．開発ホストとターゲット・ボードをRS-232-Cシリアル・ケーブル(ストレート・ケーブル)で接続し，ツールの使用方法に従って書き込みます.(ターゲット・ボードとのシリアル接続では，ほとんどの場合，クロス・ケーブルを利用するが，このボードではストレート・ケーブルを利用するので注意)．なお，ROMに書き込むことを一般に「ROMに焼く」ともいうので覚えておいてください．

アプリケーションのコーディング，ビルド，デバッグ

● コーディング

開発環境が整ったところで，組み込みシステムの定番であるLEDの点滅を行う単純なアプリケーション・プログラムを作成してみましょう(リスト1)．このプログラムは，H8/3069Fプロセッサの4番目のポートの6ビット目，7ビット目に接続されているマザーボード上の赤と緑のLEDを交互に点灯するプログラムです．プログラムの流れは，以下のとおりです．

1) ポート4の6ビット目，7ビット目を出力ポートとして設定
2) 7ビット目に1を出力
3) 点灯を保持
4) 6ビット目に1を出力
5) 点灯を保持
6) 2)に戻る

● ビルド

それでは，早速コンパイルしてみましょう(コンパイラを起動する前に，GNU開発環境へのPATHの設定を忘れないこと)．ターゲットに適したオプション，適したリンカ・スクリプト，適したスタートアップ・スクリプトを与えると，gccは

コンパイルとリンクを一度に行ってくれるので非常に便利です．ここでは，リンカ・スクリプトとしてram3069F.xを，スタートアップ・スクリプトとしてramcrt0.S（リスト2）を作成します．これらの元となるファイルはボードとともに提供されるCD-ROMの中に用意されているので，それらを流用してリスト3のように編集してください．そして，図14のようにコンパイル，リンクします．各オプションの意味は以下のとおりです．

-g ：デバッグ情報を付加する

-mh：H8/300Hシリーズのプロセッサ向けにコンパイルを行う

-mint32：32ビットのint型を使用する（デフォルトは16ビット）

-nostartfiles：標準のスタートアップ・スクリプト（crt0.S）を使用しない．

-T：リンカ・スクリプトを指定

-o：出力ファイル名の指定

● ターゲットへのダウンロードと実行

本来であれば，実機でデバッグを行う前にここでシミュレータを使用してデバッグを行います．しかし，GDBに含まれているシミュレータでは，I/Oポートなど外部のデバイスを制御するようなプログラムをデバッグすることはできません．よって，実機でのデバッグを行うことになります．作成したELF形式のバイナリ・イメージをターゲットにダウンロードするためにGDBを利用します．RedBootをROMに焼いた場合は，ターゲット・ボードがEthernetケーブルでネットワークに接続されており，IPアドレスを取得できていることを確認し，図15や図16のように実行してみてください．

● ターゲット上でのデバッグ

どうでしょうか？　うまく動作したでしょうか？　万が一うまく動作しなくても心配することはありません．そのためのデバッガです．ここで使用している各コマンドの意味は以下のとおりです．

リスト2　RAM用スタートアップ・スクリプト

```
    .h8300h
    .section .text
    .global _start
_start:
    jsr @_main
    rts
```

リスト3　RAM用リンカ・スクリプト

```
OUTPUT_FORMAT("elf32-h8300")
OUTPUT_ARCH(h8300h)
ENTRY("_start")
MEMORY
{
    ram(rwx)    : o = 0xffbf20, l = 0x1800
}
SECTIONS
{
.text : {
    *(.text)
    *(.strings)
    *(.rodata)
     _etext = . ;
    }  > ram
.tors : {
    ___ctors = . ;
    *(.ctors)
    ___ctors_end = . ;
    ___dtors = . ;
    *(.dtors)
    ___dtors_end = . ;
    }  > ram
.data : {
    *(.data)
     _edata = . ;
    }  > ram
.bss : {
     _bss_start = . ;
    *(.bss)
    *(COMMON)
     _end = . ;
    }  >ram
}
```

```
$ export PATH=/usr/local/gnu/bin:$PATH
$ h8300-elf-gcc -g -mh -mint32 -nostartfiles
-Tram3069f.x -o blink.elf ramcrt0.S blink.c
```

図14　コンパイルとリンク

```
.$ h8300-elf-gdb blink.elf
GNU gdb 6.0
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=h8300-elf"...
(gdb) target remote 192.168.1.15:9000
Remote debugging using 192.168.1.15:9000
0x000081de in ?? ()
(gdb) load
Loading section .text, size 0x6e lma 0xffbf20
Start address 0xffbf20, load size 110
Transfer rate: 880 bits/sec, 55 bytes/write.
(gdb) c
Continuing.
```

図15　ターゲットへのダウンロードと実行

```
(gdb) target remote 192.168.1.15:9000
Remote debugging using 192.168.1.15:9000
0x000081de in ?? ()
(gdb) load
Loading section .text, size 0x6e lma 0xffbf20
Start address 0xffbf20, load size 110
Transfer rate: 880 bits/sec, 55 bytes/write.
(gdb) list
6               P4DDR = 0xc0;
                  // Set bit 6 and 7 of port No.4 for Output
7               while(1){
8                       P4DR = 0x80;
                          // Write bit 7 to turn on Green LED
9                       i=1000000;
10                      while(i>0) i--; // Wait for a moment
11                      P4DR = 0x40;
                            // Write bit 6 to turn on Red LED
12                      i=1000000;
13                      while(i>0) i--; // Wait for a moment
14              }
15      }
(gdb) break main
Breakpoint 1 at 0xffbf2e: file blink.c, line 6.
(gdb) info break
Num Type           Disp Enb Address    What
1   breakpoint     keep y   0x00ffbf2e in main at blink.c:6
(gdb) c
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0x00ffbf30 in main () at blink.c:6
6               P4DDR = 0xc0;
                  // Set bit 6 and 7 of port No.4 for Output
(gdb) step
11                      P4DR = 0x40;
                            // Write bit 6 to turn on Red LED
(gdb) info registers
r0             0x00000000  0
r1             0x0000ddd1  56785
r2             0x000081d6  33238
r3             0x00000000  0
r4             0x0000ddd4  56788
r5             0x0000ddd5  56789
r6             0x005f3fec  6242284
sp             0x005f3fdc  6242268
ccr            0x00        0    I-0 UI-0 H-0 U-0 N-0 Z-0 V-0 C-0
                                              u> u>= != >= >

pc             0x00ffbf62  16760674
cycles         0x00000000  0
tick           0x00000000  0
inst           0x00000000  0
(gdb) backtrace
#0  main () at blink.c:11
```

図16　ターゲット上でのデバッグ

リスト 4　ROM用スタートアップ・スクリプト

```
    .h8300h
    .section .text
    .global _start
_start:
    mov.l   #_stack,sp
    mov.l   #___dtors_end,er0
    mov.l   #___data,er1
    mov.l   #_edata,er2
.mvdata:
    mov.w   @er0,r3
    mov.w   r3,@er1
    adds    #2,er0
    adds    #2,er1
    cmp.l   er2,er1
    blo .mvdata
    jsr @_main
    bra _start

    .section .stack
_stack: .long   1
```

target：ターゲットとの接続（シリアル接続の場合は，target remote com1）
load　：ターゲットへのダウンロード
list　：リストの表示
break ：ブレーク・ポイントの設定
info break：ブレーク・ポイントの表示
c　　 ：実行
step　：ステップ実行
info registers：レジスタの値の表示
backtrace：スタックの表示

● ROM化

作成したLED点灯プログラムは，無事に動作したでしょうか？　それができたら，次はROM化です．一般的に，ROM化とは，作成したアプリケーション・プログラムをROMに書き込み，電源オンですぐに動作を始めるように設定することを指します．よって，ROM化のための作業がまた必要になります．といっても，たいした作業ではなく，ROM用のリンカ・スク

リスト 5　ROM用リンカ・スクリプト

```
OUTPUT_FORMAT("elf32-h8300")
OUTPUT_ARCH(h8300h)
ENTRY("_start")
MEMORY
{
    rom(rx)     : o = 0x00b000, l = 0x055000
    ram(rwx)    : o = 0xffdd20, l = 0x001800
    stack(rw)   : o = 0xffff1c, l = 0x000004
}
SECTIONS
{
.text : {
    *(.text)
    *(.strings)
    *(.rodata)
     _etext = . ;
    } > rom
.tors : {
    ___ctors = . ;
    *(.ctors)
    ___ctors_end = . ;
    ___dtors = . ;
    *(.dtors)
    ___dtors_end = . ;
    }  > rom
.data : AT ( ADDR(.tors) + SIZEOF(.tors) ){
    ___data = . ;
    *(.data)
    *(.tiny)
     _edata = .;
    } > ram
.bss : {
     _bss_start = . ;
    *(.bss)
    *(COMMON)
     _end = . ;
    }  >ram
.stack : {
     _stack = . ;
    *(.stack)
    }  > stack
}
```

リプト(リスト5)とスタートアップ・スクリプト(リスト4)をリンクし，ROM用のファイル形式に変換するのみです．

図17のようにコンパイル，リンクを行います．ここで-Oオプションは最適化を行うオプションです．なぜ最適化を行うかというと，ROMのサイズが限られていること，デバッグ情報などの余計なシンボル情報は不要であることがあげられます．そして，最後にSレコード形式と呼ばれるファイル形式に変換します．

Sレコード形式とは，米Motorola社によって開発されたテキスト形式のファイルで，ROMアドレスとバイナリ・コードが規則正しく並んでいます(リスト6)．また，ほとんどのROMライタがこのSレコード形式に対応しています．

変換ができたら，GDBスタブやRedBootを焼いたときと同じようにフラッシュ・メモリに焼きます．もちろん，上書きするので以前の内容は消えてしまいます．

● パワーオン・リセット！

それでは，電源を入れてみましょう．どうでしょうか？　無事LEDが点滅したでしょうか？　もっとLEDの数を増やせば，クリスマス・イルミネーションが自作できますね．

ちなみに，電源投入後に実行されるリセットのことをパワーオン・リセットといい，どこのアドレスから実行を開始するかなどは，ベクタ・テーブルと呼ばれるテーブルで管理されています．

```
$ h8300-elf-gcc -O -mh -mint32
        -nostartfiles -Trom3069f.x
        -o blink.elf 30xxcrt0.S blink.c
```

図17　ROM化のためのコンパイルとリンク

```
$ h8300-elf-objcopy -O srec blink.elf blink.mot
```

図18　Sレコード形式への変換

リスト6　ROM用Sレコード

```
S00C0000626C696E6B2E6D6F7465
S113B0007A0700FFFF1C7A000000B0607A0100FF9D
S113B010DD207A0200FFDD20690369930B800B8138
S113B0201FA145F45E00B02A40D601006DF60FF66C
S113B030FAC06AAA00FEE0037A0300FFFFD3FA8095
S113B04068BA7A02000F42401B020FA24EFAFA407D
S113B05068BA7A02000F42401B020FA24EFA40DE89
S903B0004C
```

おわりに

誌面のつごう上，割り込み処理など，ここでは説明できなかった基礎知識もまだまだたくさんありますが，それらについてはボードの仕様書やプロセッサのデータ・シートなどを自分で読んで，実際に試してみることにより身につけて欲しいと思います．ここまでできれば，今日から皆さんも組み込み技術者の仲間入りです．

日本の組み込み情報のWebサイト(http://www.embedded.jp/)では，日本の組み込み技術者のための技術情報を提供するとともに，組み込み技術者の皆さんが交流するための各種メーリングリストも運営されています．ほかにもこのようなコミュニティはたくさん存在するので，ぜひとも積極的に交流し，新しい技術や知識を身につけていただきたいと考えます．組み込み業界に新しく入られた皆さんの検討を祈ります．

参考文献

(1) John R.Levine著，榊原一矢監訳；Linkers & Loaders,(株)オーム社，2001年9月25日第1版第1刷，ISBN4-274-06437-9
(2) 鹿取祐二著；C言語でH8マイコンを使いこなす，(株)オーム社，2003年10月10日第1版第1刷，ISBN4-274-07964-3
(3) 杉浦英樹ほか著；うまくいく！組み込み機器の開発手法，Interface，2003年5月号，CQ出版(株)
(4) 中森章著；マイクロプロセッサ技術の基本，Interface，2003年11月号，CQ出版(株)
(5) 中島信行著；Cプログラミングの基礎知識，Interface，2004年3月号，CQ出版(株)
(6)(社)日本システムハウス協会エンベデッド技術者育成委員会；組込みシステム開発のためのエンベデッド技術，(株)電波新聞社，2003年11月15日第1版第1刷，ISBN4-88554-754-7
(7) 橋本隆成著；プロジェクト管理　成功するソフトウェア開発の最新スタイル，(株)技術評論社，2004年1月9日初版第1刷，ISBN4-7741-1924-5
(8) 米Free Software Foundation(http://www.fsf.org/)
(9) Cygwinプロジェクト(http://cygwin.com/)
(10) eCos/RedBoot for H8/300プロジェクト(http://sourceforge.jp/projects/ecos-h8/)
(11)(株)秋月電子通商(http://akizukidenshi.com/)
(12) 日本の組み込み情報(http://www.embedded.jp/)

なかむら・けんいち　アップウィンドテクノロジー・インコーポレイテッド

5. CodeWarriorを使用した組み込み開発

統合開発環境を用いた組み込み開発の事例

深瀬 茂寛

かつて，組み込みソフトウェア開発では，コマンド・ライン・ベースの開発環境が用いられていた．コマンド・ラインは，習熟すればGUIなどよりも効率的な開発が行えるという利点があるが，現在のデスクトップOS向けの開発環境ではGUIベースのものが主流である．そのため，組み込み機器開発ではコマンドを学習する必要があり，組み込み系でもGUIベースの開発環境が望まれていた．

今回，もともとはデスクトップ向けソフトウェア開発に使われていたGUIベースの開発環境CodeWarriorが組み込みソフトウェア開発にも対応し，デスクトップ向けソフトウェア開発と遜色ない開発環境が提供されることになった．そこで本章では，CodeWarriorを使った組み込み開発について，実例を挙げて解説する． （編集部）

はじめに

メトロワークス（株）が開発，販売しているCodeWarriorはコンパイラやデバッガなどのツール類を含み，コード編集，ビルドとデバッグ環境が一つのGUIに統合されている統合開発環境（IDE）です．もともと，CodeWarriorはデスクトップ・コンピュータ向けの開発ツールとして始まりましたが，現在ではさまざまな組み込み機器向けソフトウェアをクロス開発するためのツールとしても数多くの実績があります．

そこで本章では，ターゲットOSにLinuxを使用したアプリケーションを開発する場合を例にとり，CodeWarriorを使用した開発方法を紹介します．

筆者がIDEを使用する理由は，大きく分けて次の3点だと思っています．

1）操作がメニューにそろっているので親しみやすい
2）定義の参照やデバッグといった各種操作を楽にしてくれる
3）操作に慣れると捨て難い

組み込みソフトウェアの開発であるかどうかにかかわらず，「IDEをどうしても使わなければならない」という必然性はありません．それ以前に，どれくらい便利なのかが事前にわからないと選択のしようがありません．筆者はCodeWarriorに慣れ親しんでいるわけではないので，本稿で使用してみた感じを読みとってください．

使用する環境

本記事を執筆するにあたり，メトロワークス（株）よりCodeWarriorとともに以下のリファレンス用のハードウェアを借りることができたので，これを使用します．

- Motorola Dragonball MX1
- COMPAQ iPAQ Pocket PC H3600

Dragonball MX1はARMプロセッサの評価ボード（写真1），iPAQは市販されていたPDAで，両者は表1のような仕様になっています．

開発ツールにはCodeWarrior製品群のうち，Linuxアプリケーション開発に対応している，

- CodeWarrior Development Studio Embedded Linux Application Limited Edition（以下，CW eLinux LE）

を使用します．

近年の組み込み機器では，プロセッサの処理能力の向上と機

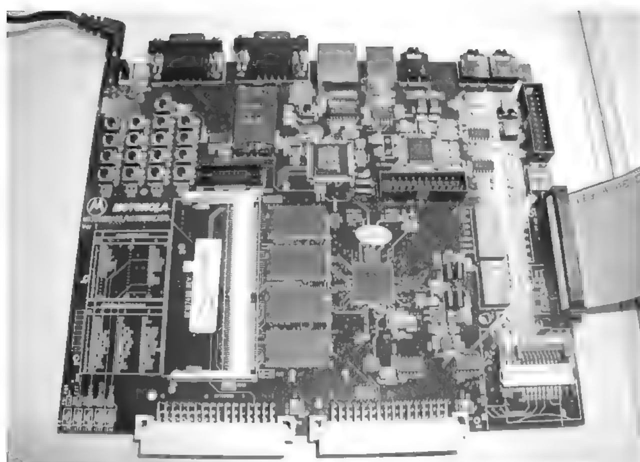


写真1 ARMプロセッサ評価ボードDragonball MX1（Motorola）

表1 COMPAQ iPAQ Pocket PC H3600とDragonball MX1の仕様

	iPAQ H3630	Dragonball MX1
プロセッサ	Intel StrongARM-1110（32ビット RISC 206MHz）	ARM920T（32ビット RISC 150-200MHz）
RAM	32Mバイト SDRAM	32Mバイト SDRAM
そのほか	ジャケット＋Ethernet PC Card	オンボードEthernet，RS-232-C

COLUMN

カーネルとドライバのデバッグについて

本文ではアプリケーションのデバッグについて記しましたが，CodeWarrior for Embedded Linux Kernel Debug Edition（以下，Kernel Debug Edition）を使用した場合のカーネルとドライバのデバッグについて解説します．

アプリケーションのデバッグは，カーネルを経由して特定のプロセスだけを実行制御し，そのほかのプロセスは影響を受けずに動作しています．つまり，デバッガでBreakしている間もターゲットのCPUは停止していません（これをRun Mode Debuggingと呼ぶ）．

これに対してカーネルをBreakpointで停止させるには，CPUを停止するしくみ（これをStop Mode Debuggingと呼ぶ）が必要になります．Kernel Debug Editionでは，JTAGデバッガと連動する方法となっています．そのため，

- JTAGデバッグに対応したCPUであること
- JTAGコネクタを持つ（CPUから信号を引き出してある）ボードであること
- JTAGデバッガがCodeWarriorとの連動に対応していること

が条件となります．

開発環境のハードウェア構成は，図Aのようになります．今回使用した機材を例にすると，ARM社のMulti-ICEというJTAGデバッガをDragonball MX1ボードで使用することが可能です．

Kernel Debug Editionのパッケージには，以下のモジュールが追加されています．

- JTAGコントロール用のDLL
- Linux Kernel Aware DLL

前者がJTAGデバッガと連動してStop Mode Debuggingを実現し，後者がLinuxカーネルが使っているMMUの情報とカーネルの情報を制御します．

これらにより，以下の機能が実現されています．

- カーネル内でのBREAK，ソース・コード・デバッグ
- 論理アドレス→物理アドレスの変換
- カーネル・スレッド情報の表示
- insmodコマンドでロードされたデバイス・ドライバのデバッグ

● カーネルのコンパイル

アプリケーションのビルドでは，CodeWarriorプロジェクトを使いましたが，残念ながらカーネルのコンパイルにCodeWarriorを使用することはできません．CodeWarriorはデバッガとして動作します．Linuxホスト上で通常のカーネル再構築と同様の手順でコンパイルし，その際にデバッグ・オプション（-g）をつけておきます．

できあがったカーネルのELFファイルをCodeWarrior IDEで使用します．

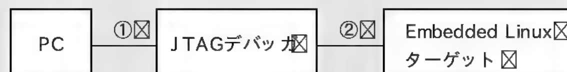
●デバッグの手順

CodeWarrior IDEにカーネルのELFファイルをドラッグ＆ドロップすると，カーネルがメモリに転送され，デバッグ情報がCodeWarriorに読み込まれます．つまり，この時点でターゲットのメモリが読み書き可能な状態になっている必要があります．通常，JTAGデバッガではこのような基本的なハードウェアの初期化はデバッガ起動時のターゲット初期化スクリプトなどの設定で行いますが，Linuxでは一般的にブート・ローダで行われます．これらの初期化はブート・ローダを一度動作させておくだけで済みます．

CodeWarriorからカーネルを転送する際には，カーネル起動パラメータが設定できるようになっているので，ブート・ローダがカーネルをメモリに転送する代わりにCodeWarriorが転送を行うこと以外に違いはありません．

メモリへの転送が終了すると，カーネルのスタート・アドレスにプログラム・カウンタが設定されるので，実行すれば通常と同じ動作になります．

これでカーネルとスタティックにリンクされたデバイス・ドライバは，この状態でデバッグ可能ですが，ダイナミックにロードされるデバイス・ドライバはinsmodコマンドでデバイス・ドライバをロードした後からデバッグが可能になります．



①: JTAGデバッガによって違う（USB，Ethernet，パラレルが主流）
②: JTAG

図A　開発環境のハードウェア構成

器に求められる機能の増加により，ターゲットにOSを搭載する例が増えており，その際にLinuxを採用することもあると思います．本稿ではLinuxそのものの移植やデバイス・ドライバの開発ではなく，ユーザ・アプリケーションの開発のみを扱い，ターゲットではLinuxがすでに動作しているものとします．Linuxカーネルをターゲットにインストールするには，同じくメトロワークス（株）より販売されているPlatform Creation Suite for Linux OSという別のソフトウェアが必要です．カーネルとドライバのデバッグも行いたい場合には，CodeWarriorの上位製品であるKernel Debug Editionを使用すれば可能になります．

CodeWarrior Development Studio Embedded Linux Application Limited Edition について

CW eLinux LEは，ターゲットのOSにLinuxを用いる場合のアプリケーション開発用としてカスタマイズされたCode Warrior製品の一つで，Windows上で動作します．PCとターゲットをEthernetまたはシリアルで接続し，Windows上のIDEが独自のプロトコルを使用してターゲットとのやりとりを行うリモート・デバッグという方法をとります．

● コンパイル，そしてリンク

コンパイラ，リンカなどのオブジェクト生成にはCygwin環境用のgccが用いられ，IDEメニューからメイクすると自動的に呼び出されます．そのため，Cygwinをインストールしておく必要があります．CygwinはWindows上でUNIXのアプリケーションを動作させるためのオープン・ソース・ソフトウェアです．

ここでひとつ問題があります．Cygwin環境は`Cygwin1.dll`を必要としますが，CW eLinux LEに含まれている`Cygwin1.dll`とバージョンが異なったものがインストールされているとコンパイルに失敗してしまいます．そのため，CW eLinux LEに含まれている`Cygwin1.dll`が用いられるよう，ほかのファイルは名前を変更するなどして無効にしておく必要があります．

● ターゲットへのダウンロード/実行/デバッグ

実行，デバッグを行うためには，先にターゲット上でMetroTRKというプログラムを起動する必要があります．MetroTRKは図1に示すようにターゲット上の常駐サービスのようなもので，CodeWarrior IDEとの通信を行うプログラムです．ターゲット上でのアプリケーションの制御，ファイル・システムへの書き込みなど，CodeWarrior IDEから行う操作はすべてMetroTRKを経由して行われます．今回はハードウェアを2機種使用しましたが，ターゲットのIPアドレス以外に使用方法に違いはありませんでした．

ここから先は，IDEメニューからDebugを選ぶだけでターゲットへ転送，起動してエントリ・ポイントで停止するところまでやってくれるので，Windows用のソフトウェア開発のような感覚です．以下，実際のプログラムを使って説明します．

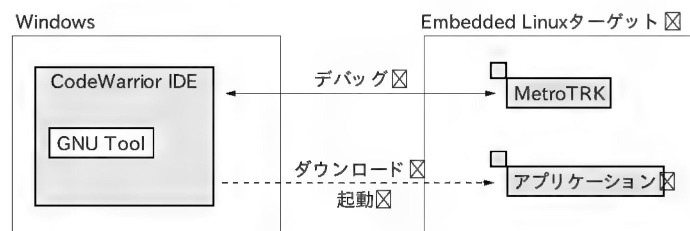


図1　CodeWarrior IDEとの通信を行うプログラムMetroTRK

CodeWarriorを使用したプログラム開発

● 単純なWelcomeアプリケーション

アプリケーションのひな型はIDEが自動的に作成してくれます．IDEメニューより File→New を選択し，Linux RAD Stationary Wizardでプロジェクトを作成します．OutputTypeはApplication，LanguagesはC，C&CPP，CPPから選べますが今回はCPPを選択しました．これで単純な「Welcome」アプリケーションができあがります（リスト1）．IDEメニューのProject→Runを選択するとターゲットにプログラムが転送され注1，デバッガで実行されて`main`関数で止まります注2．

デバッグ・ウィンドウでステップ操作をしていくと，図2のようにMetroTRK Consoleに“Welcome”と表示されます．通常のLinuxプログラムは，OSによって適切なデバイス（tty）が選択されてCRTやシリアルに出力されますが，この動作は

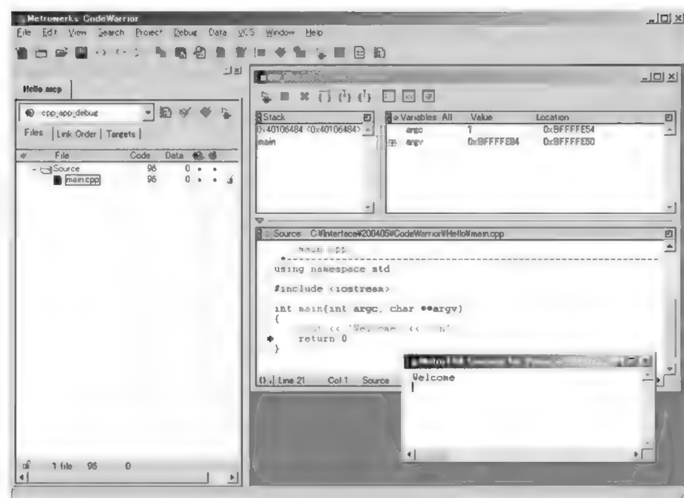


図2　MetroTRK Consoleに“Welcome”と表示

リスト1　IDEが自動的に作成した「Welcome」アプリケーション

```
/*
 *  Copyright (c) 2001-2002 Metrowerks Corporation.
 *   All Rights Reserved.
 *
 *  Questions and comments to:
 *        <mailto:support@metrowerks.com>
 *        <http://www.metrowerks.com/>
 */


/*-------------------------------------------------------*
    main.cpp
 *-------------------------------------------------------*/
using namespace std;

#include <iostream>

int main(int argc, char **argv)
{
    cout << "Welcome" << "¥n";
    return 0;
}
```

注1：転送される先はIDEメニューのEdit→cpp_app_debug Settingsでダイアログを開き，Debugger→Remote Debugging→Remote Download Pathにより変更できる．かならずターゲットのファイル・システムにパスが存在している場所を指定しなければならない．

注2：`main()`関数以外の任意の関数で止めることやプログラムのエントリ・ポイントで止めることもできるが，説明は割愛した．

注3：デバッグ・ウィンドウへの出力が，デバッガとターゲットとの通信を妨げることがある．無限ループ内などで出力するとバッファリングが追いつかなくなってデバッガの操作ができなくなることがある．

MetroTRK がアプリケーションのI/Oを横取りしてデバッガのウィンドウに出力しているわけです．この方法は，ターゲットにコンソール出力用のデバイスが何もないような場合にも使えるので便利です．IDE メニューの Edit → cpp_app_debug Settings でダイアログを開き，Debugger → Console I/O Settings によってファイル/デバッガのウィンドウ/OSのコンソールの三つのうちから選んで設定することができます[注3]．

● ステーショナリ機能で任意のひな型を登録

中には「なぜ最初のプログラムが（リスト 2のように）Hello Worldじゃないんだ？」と思った読者もいるのではないでしょうか．そのようなときには，ステーショナリ機能を使うことで任意のひな型を登録することができます．ステーショナリの登録は，以下の手順で行います．

1) ひな型にするプロジェクトを開く
2) メニューから File → Save A Copy As を選択
3) CodeWarrior のステーショナリ・ディレクトリ（通常は C:¥Program Files¥Metrowerks¥CodeWarrior¥Stationery）以下にフォルダを一つ作り，その中に保存する
4) ソース・ファイルはエクスプローラなどで 3)のフォルダへコピーする

文章で書くと煩雑なように思われるかもしれませんが，実際には図3（a)のようにコピーしただけです．次回プロジェクトの新規作成を行う際には，図3（b)のようにメニューに現れます．しくみはシンプルですが，プロジェクト・ファイルの中にはターゲットの設定やコンパイラ/リンカのオプションなども含まれているので，同じ設定のアプリケーションを複数作りたいときなどに便利でしょう．単にプロジェクト・ファイルをコピーするだけではなく，新しいプロジェクト名で作ることができるところがミソです．

● 伝言ゲーム・アプリケーションのプログラム

少しだけ複雑な動きをデバッグするために，リスト 3のプログラムを作ってみました．このプログラムは pthread ライブラ

リスト 2　おなじみのプログラム Hello World

```
//
// First program must be printout "Hello World".
//
using namespace std;

#include <iostream>

int main(int argc, char **argv)
{
    cout << "Hello World" << endl;
    return 0;
}
```

COLUMN 2

print 文について

Linux（UNIX)では，標準入出力が当たり前に使えるものとして扱われていると思いますが，組み込み機器の開発においては最終製品に文字出力を行うためのデバイスが存在せず，print 文（C標準ライブラリの printf や C++ 標準ライブラリの cin/cout/cerr)は製品では使用しない，使用できない，ということが多いと思います．

そのような場合，print 文の使用方法はユーザとの対話などではなく，ログ出力などデバッグ目的に限定されてきます．

実際の方法としては，

A）開発途中のボードにはシリアル・デバイス（UART)を搭載する
B）CodeWarrior のようなデバッガの出力を利用する
C）メモリ上のバッファに書き込んでおき，ダンプして見る

といったところでしょう．

Linux カーネルは，再構築でコンソールI/Oをシリアル・デバイスにすることができるので，A)の採用は簡単です．B)の方法は UART が取り除かれたボードでも使用することができるという点でメリットがあります．C)は標準ライブラリ関数ではできませんが，Linux ならばファイルに出力することで，ほぼ同等のことができます．

デバッグが済んでしまうと，今度は print 文が邪魔になってくると思います．Linux の場合は出力先を"/dev/null"に変えることで何も出力されない状態にすることは可能です．この方法ではコンパイルされたオブジェクトには print 文は残ってしまっているのでむだが多くあります．残念ながらコンパイラ/CodeWarrior/カーネル/そのほかツールどれにも print 文だけを完全に取り除くという機能はないので，昔ながらのプログラミングのテクニックで対応することになります．

たとえば C標準ライブラリの printf であれば，

```
#ifdef DEBUG
    #define DEBUG_PRINTF(x) printf x
#else
    #define DEBUG_PRINFT(x)
#endif
```

といった定義をしておいて

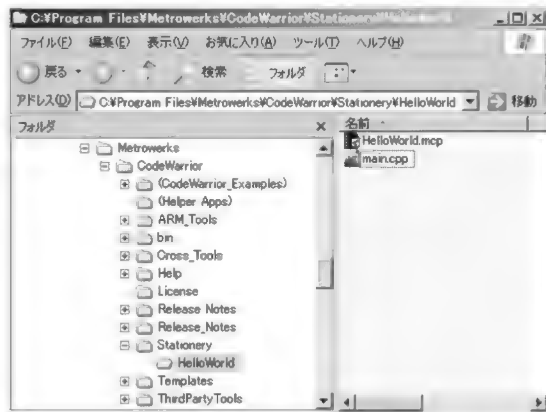
```
DEBUG_PRINTF(( "This is %dth debug message.¥n",
                                          n ));
```

のような書きかたをしておくと，リリース用のビルドからは printf 文が除外されてオブジェクト・サイズも小さくできるのでよく行う手法だと思います．

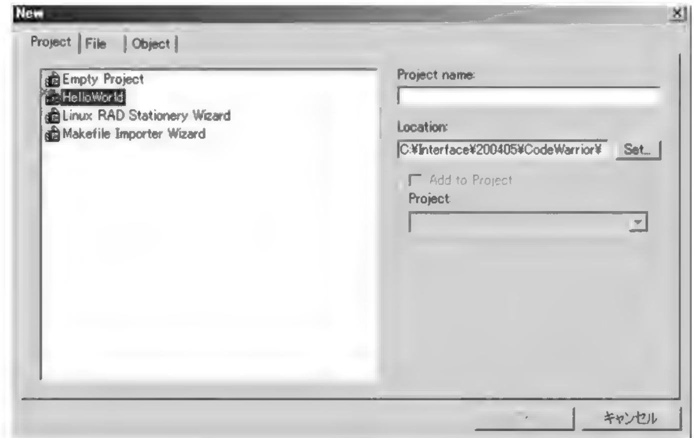
print 文はよく使う機能でありながら，組み込みの開発においては以下の理由でまず最初に print 文の機能を独自に実装しなければならないことが多々ありました．

- コンパイラに付属の標準ライブラリ関数がスレッドに対応していない（≒リエントラントでない)ことがある
- ターゲット上にファイル・システムがない
- 文字出力のためのデバイスがないか，特殊なもの

製品には搭載されない機能に手間と時間を割くということは，できればなくしたい作業です．Linux+CodeWarrior の開発環境ではうまくカバーされていると思います．



（a）ステーショナリ機能の登録



（b）プロジェクトの新規作成

図3　ステーショナリ機能を使う例

リを用いてスレッドを作り，スレッド間で伝言ゲームをします（コンピュータは聞き違えることはないのでゲームとはいえないが）．動作を簡単に説明すると，

main（親）：
- 子スレッドを四つ（ts，t1，t2，t3）作る
- 10秒スリープする
- シグナル（SIGUSR1）をtsスレッドへ送信
- 子がすべて終了するまで待つ（join）

Thread1～3：
- Thread1の入力は標準入力，th3の出力は標準出力
- Thread1→Thread2はUNIXドメイン・ソケットで伝言
- Thread2→Thread3はpipeで伝言

ThreadSig：
- SIGUSR1を受け取ったらファイル・ディスクリプタを閉じる（Thread1～3が終了する）

となり，実行すると約10秒の間にタイプ・コンソールにタイプされた文字をエコー・バックする動作になります．このプログラムはビルド/デバッグする前に二つほど設定を行う必要があります．

● ライブラリの設定

リスト3はpthreadライブラリ関数を用いているので，pthreadライブラリをリンクする必要があります．コンパイラやリンカなどGNUのコマンドに対する設定はコマンド・ライン引き数をそのままテキスト・ボックスに記入する形式になっています．そのため，GUIの使いやすさはありませんが，設定できないオプションもないと思います．ここではcpp_app_debug

リスト3　pthreadを使った伝言ゲーム・プログラム

```
/*****************************************************************
 * Message Sample
 *****************************************************************/
using namespace std;

#include <iostream>

#include <stdio.h>
#include <fcntl.h>      /* open */
#include <unistd.h>     /* sleep, pipe, */
#include <sys/wait.h>   /* SIGUSR1 */
#include <pthread.h>
#include <sys/socket.h> /* socket, accept, connect, ... */
#include <sys/un.h>     /* struct sockaddr_un */

void Thread1( int* );
void Thread2( int* );
void Thread3( int* );
void ThreadSig( int* );

void die( char* msg, int code );
int  copy( int from, int to );

pthread_t ts, t1, t2, t3;
int rs = 0, r1 = 0, r2 = 0, r3 = 0;

#define SOCKNAME "sample-uds"
```

```
int iofd[2];
int pipefd[2];
int sockfd[2];

/*-------------------------------------------------------------*
    main
 *-------------------------------------------------------------*/
int main( int argc, char** argv )
{
    cout.width(0);
    cout << "Hello World" << "¥n";

    iofd[0] = STDIN_FILENO;
    iofd[1] = STDOUT_FILENO;

    if( pipe( pipefd ) == -1 ) die( "Create pipe", 1 );

    unlink( SOCKNAME );

    pthread_create( &ts, NULL, (void*(*)(void*)) ThreadSig,
                    (void*) &rs );
    pthread_create( &t1, NULL, (void*(*)(void*)) Thread1,
                    (void*) &r1 );
    pthread_create( &t2, NULL, (void*(*)(void*)) Thread2,
                    (void*) &r2 );
    pthread_create( &t3, NULL, (void*(*)(void*)) Thread3,
                    (void*) &r3 );
```

リスト 3　pthreadを使った伝言ゲーム・プログラム(つづき)

```
    sleep(10);

    cout << "Main timeup. Send signal " << SIGUSR1
         << " to thread " << t3 << "." << endl;
    pthread_kill( ts, SIGUSR1 );

    pthread_join( t1, NULL );
    pthread_join( t2, NULL );
    pthread_join( t3, NULL );

    cout << "All done"
         << ", r1=" << r1 << ", r2=" << r2 << ", r3=" << r3
         << endl;

    return 0;
}

/*-------------------------------------------------------------*
 *  thread functions
 *-------------------------------------------------------------*/
void Thread1( int* parg )
{
    struct sockaddr_un addr;
    size_t alen;

    if( ( sockfd[1] = socket( PF_UNIX, SOCK_STREAM, 0 ) ) < 0){
        die( "socket", sockfd[1] );
    }
    addr.sun_family = AF_UNIX;
    strcpy( addr.sun_path, SOCKNAME );
    alen = sizeof( addr.sun_family ) + strlen( addr.sun_path );
    for( ;; ) {
        if( connect( sockfd[1], (struct sockaddr*) &addr, alen
                                                          ) ) {
            sleep(1);
        } else {
            cout << "Thread 1 connected." << endl;
            break;
        }
    }

    *parg = copy( iofd[0], sockfd[1] );
    cout << "Thread 1 done, wrote " << *parg << " bytes."
         << endl;
    close( sockfd[1] );
}

void Thread2( int* parg )
{
    struct sockaddr_un addr;
    int    conn, rcd;
    size_t alen;

    if( ( sockfd[0] = socket( PF_UNIX, SOCK_STREAM, 0 ) ) < 0){
        die ( "socket", sockfd[0] );
    }

    addr.sun_family = AF_UNIX;
    strcpy( addr.sun_path, SOCKNAME );

    alen = sizeof( addr.sun_family ) + strlen( addr.sun_path );

    if( ( rcd = bind( sockfd[0], (struct sockaddr *) &addr,
                                                  alen ) ) ) {
        die( "bind", rcd );
    }

    if( ( rcd = listen( sockfd[0], 5 ) ) ) {
        die( "listen", rcd );
    }
    conn = accept( sockfd[0], (struct sockaddr *) &addr,&alen);
    if( conn < 0 ) {
        cout << "Thread 2 done." << endl;
        die( "accept", conn );
    }
    cout << "Thread 2 getting data..." << endl;
    *parg = copy( conn, pipefd[1] );
    cout << "Thread 2 done, wrote " << *parg << " bytes."
         << endl;
    close( conn );
    close( sockfd[0] );
    unlink( SOCKNAME );
}

void Thread3( int* parg )
{
    cout << "Thread 3 getting data..." << endl;
    *parg = copy( pipefd[0], iofd[1] );
    cout << "Thread 3 done, wrote " << *parg << " bytes."
         << endl;

    close( pipefd[0] );
    close( pipefd[1] );
}

#include <signal.h>
void ThreadSig( int* parg )
{
    int num;
    sigset_t sig_block, sig_catch;

    sigemptyset( &sig_block );
    sigaddset( &sig_block, SIGUSR1 );
    pthread_sigmask( SIG_BLOCK, &sig_block, NULL );

    sigemptyset( &sig_catch );
    sigaddset( &sig_catch, SIGUSR1 );

    sigwait( &sig_catch, &num );
    cout << "Catch signal[" << num << "]." << endl;

    close( iofd[0] );
    close( pipefd[0] );
    close( pipefd[1] );
    cout << "Signal catcher thread done." << endl;

}

/*-------------------------------------------------------------*
 *  utility functions
 *-------------------------------------------------------------*/
void die( char* msg, int code )
{
    perror( msg );
    exit( code );
}

int copy( int from, int to )
{
    char buf[1024];
    int  rbytes, wbytes, total = 0;

    fd_set rfds, wfds;
    struct timeval tv;
    int rcd;

    for( ;; ) {
        FD_ZERO( &rfds );
        FD_SET( from, &rfds );

        FD_ZERO( &wfds );
        FD_SET( to, &wfds );

        tv.tv_sec = 0; tv.tv_usec = 0;
        rcd = select( from+1, &rfds, NULL, NULL, &tv );
        if( rcd > 0 ) {
            rbytes = read( from, buf, sizeof(buf) );

            rcd = select( to+1, NULL, &wfds, NULL, &tv );
            if( rcd < 0 ) break;
            if( ( wbytes = write( to, buf, rbytes ) )
                         != rbytes ) {
                die( "write", wbytes );
            }
        total += wbytes;
        } else
        if( rcd == 0 ) {
            continue;
        } else {
            break;
        }
    }
    return total;
}
```

Settings ダイアログの，Linker → GNU Linker の Libraries に「 -lpthread」と書いておきます．

● シグナル制御の設定

アプリケーション・プロセスがシグナルを受け取ったときに，

1) デバッガがそのシグナルを制御する

2) プロセスにシグナルを渡す

という制御をそれぞれ独立して設定可能になっています．1)の場合はデバッガが例外としてキャッチします．リスト 3では SIGUSR1 シグナルをアプリケーション内で処理したいので，cpp_app_debug Settings ダイアログの，Debugger → Debugger Signals により SIGUSR1 の設定を Pass on にします．

● マルチスレッド・デバッグ

リスト 3は実行中にスレッドを作るため，プログラムはその時点から複数のスレッドが並行動作することになります．CodeWarrior のデバッガはマルチスレッドに対応しているので，スレッドが作成されると，図4のようにスレッドごとにデバッグ・ウィンドウが開きます（リスト 3の場合は五つ開く）．このとき，任意の場所にブレーク・ポイントを設定しておけばスレッドごとにステップ実行することもできます．

ブレーク・ポイントはデバッガに欠かせない機能ですが，利用に適さない場合もあります．たとえばリスト 3の場合，親（main)を実行したまま子スレッドのいずれかをステップ実行したとすると，約 10秒後にはシグナルが発行されてしまいます．また，Thread1 の connect() 関数呼び出し部分はリトライしていますが，何回リトライされたかを調べたいときにブレークされてしまうと動作が大きく変わってしまいます．コンソールへの printf 文を入れておくこともできますが，そのためにはプログラム自体をコンパイルしなおさなければなりません．そのうえ，オブジェクト・サイズに厳しい組み込みソフトウェアの開発では printf 文自体が使えないこともあります．

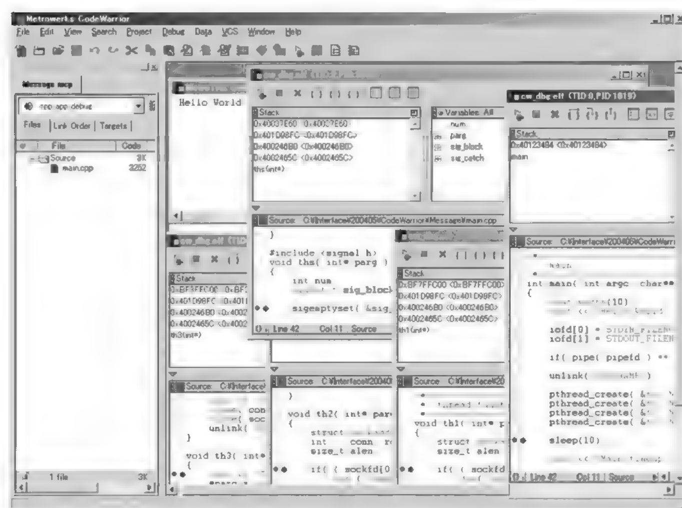


図4　スレッドごとのデバッグ・ウィンドウ

そんなときのために CodeWarrior にはイベント・ポイントという機能があります．イベント・ポイントは，プログラムが設定箇所を通過したときにデバッガにアクションを起こさせる機能で，設定できるアクションには，

- Log Point ： 通過時にファイルに出力する
- Pause Point : 通過時にデバッガの表示を更新する
- Script Point : 通過時にスクリプトやほかの Windows プログラムを実行する
- Skip Point ： 指定行の実行を行わずにスキップする
- Sound Point : 通過時に音を鳴らす

の 5種類があります．

たとえば，リスト 3の Thread2 の関数入り口にブレーク・ポイントを設定し，Thread1 の sleep() 呼び出し部分に Sound Point を設定しておくと，通過するたびにチャイムが鳴ります．

イベント・ポイント通過時にはブレーク・ポイントよりは短い時間で済みますが，ターゲット・プログラムはいったん停止するので，時間にシビアな動作や割り込みなどのタイミングに依存する動作は，実際とは変わってしまうので注意してください．プログラムをまったく変更せずにいろいろな動作を設定できるところがメリットです．

● 別のコマンドの実行

マルチスレッドと同様に，あるプログラムから別のプログラムを起動した場合のマルチプロセス・デバッグも行うことができますが，マルチプロセスのデバッグを行う際にはコーディングとデバッガの設定のそれぞれに注意しなければならない点があるので，別の例を用いて説明します．

リスト 4 a)は標準入力から入力された文字列をコマンドとして実行するプログラムです．シェルやキャラクタ・ベースの対話型のプログラムを書こうとすると，最初はだいたい似たようなものになるのではないでしょうか．

● システム・コールの置き換え

コーディングについての注意点は，現状の Linux カーネルでは fork システム・コールの中へはデバッガで追うことができず，clone システム・コールならば可能だということです．しかし，clone は Linux に固有のもので POSIX などの規格にもないものなので，移植性の面からあまり使わないほうが良いでしょう．そこでリスト 4 b)のようにマクロで置換することによって，コードを変えずにデバッガを活用することができます．また，C 標準ライブラリの system 関数を使った場合もデバッガが子プロセスを追うことができません．なおリスト 4 b)では，C 標準ライブラリに手を加えずに system 関数自体を置き換えてしまっています．そのため手軽ですが，細かな点で C 標準ライブラリと動作が異なるかもしれません．

● デバッガの設定

exec によって実行される別のプログラムの中へステップ・インするためには，デバッガが対象のプログラムのデバッグ情報とソース・コードの関連を把握している必要があります．そ

リスト 4 マルチプロセス・デバッグを行った場合のサンプル・プログラム

```
// Execute other program.

using namespace std;

#include <iostream>
#include <string>
#include <sys/wait.h>
#include <unistd.h>
#include <fcntl.h>
#include "cw_dbg.h"

#include <stdio.h>
#include <stdarg.h>

#if !defined(__CYGWIN__)
#ifndef O_BINARY
#define O_BINARY    0
#endif  // O_BINARY
#endif  // __CYGWIN__
#define O_WRITE     ( O_WRONLY | O_CREAT | O_BINARY )

#define MAXBUFSIZ   80

static char prompt[32] = "$ ";

// Argument Buffer
typedef struct argbuf_t {
    int   argc;
    char* argv[MAXBUFSIZ];
} argbuf_t;

int chop( char* s )
{
    for( ; (s && *s); s++ ) {
        if( ( *s == '¥n' ) || ( *s == '¥r' ) ) {
            *s = '¥0';
            return 1;
        }
    }
    return 0;
}

int exec_child( int argc, char* const argv[], int fd_in,
                int fd_out, int fd_err )
{
    int pid, status;
    int newin, newout, newerr;

    cerr << "I am " << getpid() << ", parent is " << getppid()
         << '.' << endl;
    pid = fork();
    if( pid < 0 ) {
        cerr << "fork error " << pid << endl;
        exit( 1 );
    } else
    if( pid == 0 ) {
        // Child process
        if( fd_in  != STDIN_FILENO  ) {
            newin = dup2( fd_in,  STDIN_FILENO  );
            if( newin == -1 ) perror( "dup2" );
        }
        if( fd_out != STDOUT_FILENO ) {
            newout = dup2( fd_out, STDOUT_FILENO );
            if( newout == -1 ) perror( "dup2" );
        }
        if( fd_err != STDERR_FILENO ) {
            newerr = dup2( fd_err, STDERR_FILENO );
            if( newerr == -1 ) perror( "dup2" );
        }
        if( fd_in != STDIN_FILENO ) { close( fd_in ); }
        if( (fd_out!=STDOUT_FILENO) && (fd_out!=STDIN_FILENO)){
            close( fd_out );
        }
        if( (fd_err!=STDERR_FILENO) && (fd_err!=STDOUT_FILENO)
              && (fd_err!=STDIN_FILENO) ) {
            close( fd_err );
        }
        execvp( argv[0], argv );
        perror( argv[0] );
        exit( 0 );
    } else {
        // Parent process
        waitpid( pid, &status, 0 );
        cerr << "Child process " << pid
             << " done with code " << WEXITSTATUS(status)
             << '.' << endl;
        return WEXITSTATUS(status);
    }
    cout << "End. I am parent. " << getpid() << ", child is "
         << pid << '.' << endl;
    return pid;
}

// Check token list.
// params:
//      c       Character for test
//      ntoken  Num of arguments
//      ap      'int' type argument list
// return:  0=false, 1=true
int isvtoken( char c, int ntoken, va_list ap )
{
    int i;
    char tok;
    for( i=0; i<ntoken; i++ ) {
            tok = va_arg( ap, int );
            if( c == tok ) return 1;
    }
    return 0;
}

// Split line buffer by token(s).
// params:
//      pargc   Destination
//      pargv   Destination
//      s       Line buffer
//      ntoken  Num of arguments
//      ...     'int' type argument list
// return:  0=success, -1=error
int split_line( int* pargc, char* pargv[], char* s, int ntoken,
                  ... )
{
    va_list ap;
    char    bflag = 0;

    if( !s || !*s ) return -1;

    va_start( ap, ntoken );
    for( ; s && *s; ) {
        if( bflag ) {
            for( ; s && *s && !isvtoken( *s, ntoken, ap );s++);
            bflag = 0;
        } else {
            for( ; s && *s && isvtoken( *s, ntoken, ap ); s++ )
                 *s = '¥0';
            if( s && *s ) pargv[ (*pargc)++ ] = s;
            bflag = 1;
        }
    }
    va_end( ap );
    return 0;
}
void printbuf( int argc, char** argv )
{
    int i;
    for( i=0; i<argc; i++ ) {
        cout << "buf[" << i << "] = ¥"" << argv[i] << "¥""
             << endl;
    }
}

int process_line( char* line )
{
    // If you want to use own standard I/O file descriptor
    //   for child process, change here.
    int    fd_in = STDIN_FILENO, fd_out = STDOUT_FILENO,
           fd_err = STDERR_FILENO;
    int    argc = 0;
    char* argv[MAXBUFSIZ];
    int    rcd  = 0;
```

（a）標準入力から入力された文字列をコマンドとして実行するプログラム

リスト 4　マルチプロセス・デバッグを行った場合のサンプル・プログラム（つづき）

```
    memset( argv, 0, sizeof(char*)*MAXBUFSIZ );

    split_line( &argc, argv, line, 2, ' ', '¥t' );
//  printbuf( argc, &argvbuf );

    if( !argc ) return 1;
    if( (argc==1) && ( string("exit")==argv[0] ) ) return 1;

    /* Execute Command. */
    if( (argc==1) &&
        (fd_in==STDIN_FILENO) && (fd_out==STDOUT_FILENO) &&
        (fd_err==STDERR_FILENO) ) {
        rcd = system( argv[0] );
    } else {
        rcd = exec_child( argc, argv, fd_in, fd_out, fd_err );
    }
    // If you use own standard I/O file descriptor, close here.
    if( fd_in  != STDIN_FILENO  ) close( fd_in  );
    if( fd_out != STDOUT_FILENO ) close( fd_out );
    if( fd_err != STDERR_FILENO ) close( fd_err );
    cout << endl;
    return 0;
}

int main( int argc, char** argv )
{
    int  status = 0;
    char linebuf[MAXBUFSIZ];

    for( ; status==0; ) {
        cout << prompt;
        memset( linebuf, 0, sizeof(char)*MAXBUFSIZ );
        if( fgets( linebuf, sizeof(linebuf), stdin ) == NULL )
            break;
        if( strlen(linebuf) == 1 ) continue;
        chop( linebuf );
        status = process_line( linebuf );
    }
    cout << "status = " << status << endl;
    cout << endl;
    return 0;
}
```

（a）標準入力から入力された文字列をコマンドとして実行するプログラム（つづき）

```
#ifndef SYSTEMCALL_WRAPPER_FOR_CODEWARRIOR_DEBUGGING
#define SYSTEMCALL_WRAPPER_FOR_CODEWARRIOR_DEBUGGING

#include <asm/unistd.h>
#include <unistd.h>
#include <sched.h>
#include <signal.h>
#include <errno.h>

#define __NR___dbg_clone     __NR_clone
_syscall2( int, __dbg_clone, int, flags, int, stack );

int __dbg_fork(void)
{
    return ( __dbg_clone( SIGCHLD | CLONE_PTRACE, 0 ) );
}
extern __typeof (__dbg_fork) __fork __attribute__ (( weak,
                                        alias ("__dbg_fork") ));
#define fork    __dbg_fork

extern int exec_child( int, char* const [], int, int, int );
int __dbg_system( const char* s )
{
    int   argc = 1;
    const char* argv[2] = { s, NULL };
    return exec_child( argc, (char*const*) &argv[0],
                  STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO );
}
#define system  __dbg_system


#endif  // End of SYSTEMCALL_WRAPPER_FOR_CODEWARRIOR_DEBUGGING
```

（b）system関数をマクロで置換したもの

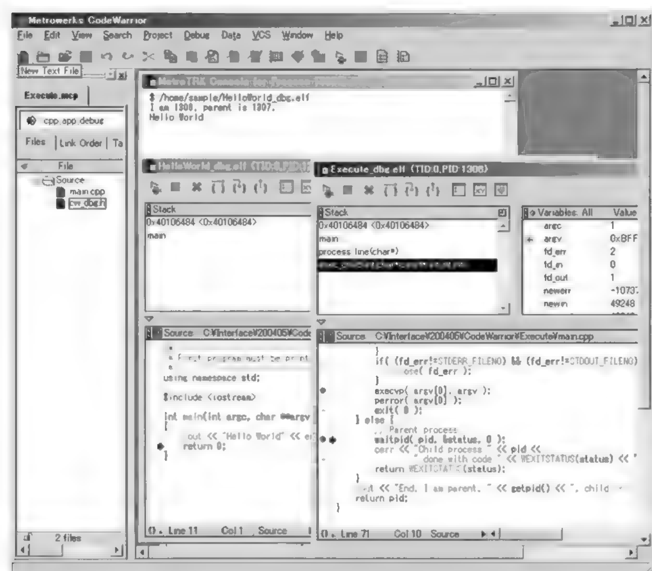


図5　リスト2の実行例

こでIDEメニューから Edit → cpp_app_debug Settings でダイアログを開き，Debugger → Other Executables にデバッガが追跡するべきコマンドの実行ファイルのパスを登録しておかなければなりません．

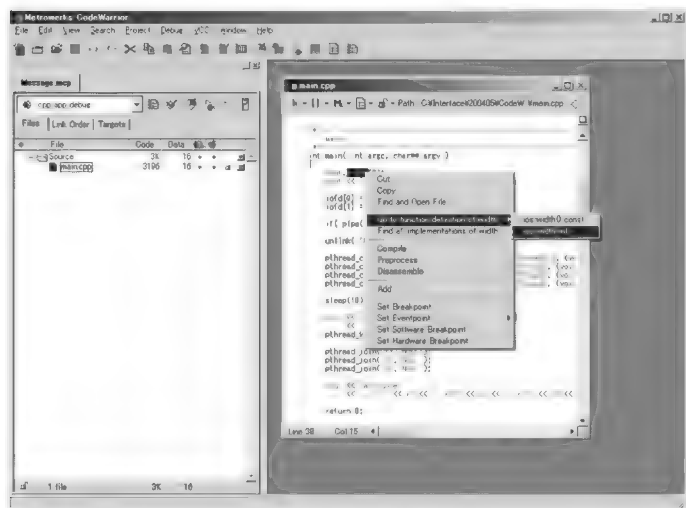
● マルチプロセス・デバッグ

リスト4を実行しコンソールに表示されるプロンプトに入力すると，forkされた時点で，マルチスレッドのデバッグと同様にデバッグ・ウィンドウが開きます．入力された文字列がデバッガに登録されているものの場合は，execが呼ばれたときに対象のプログラムのソース・コードが表示され，入り口で停止します．

図5は，リスト2を「HelloWorld_dbg.elf」として登録しておいて実行した例です．登録されていない文字列の場合には，デバッガが一度プログラムの所在をたずねてきます．キャンセルしてもプログラムの実行は続けられますが，子プロセス内をデバッグすることはできません．たとえば，コンソールから「ls」と入力してダイアログをキャンセルした場合，コマンドlsは正しく実行され，コンソールに結果が出力されますが，lsコマンド内で止めることはできないということです．

そのほかの機能

プログラミング・デバッグの話はここまでとして，CodeWarriorを使用して便利だった機能をいくつか紹介します．


（a）ソース・ブラウザ

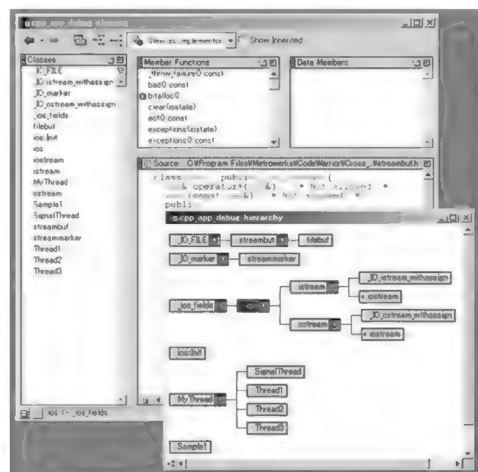
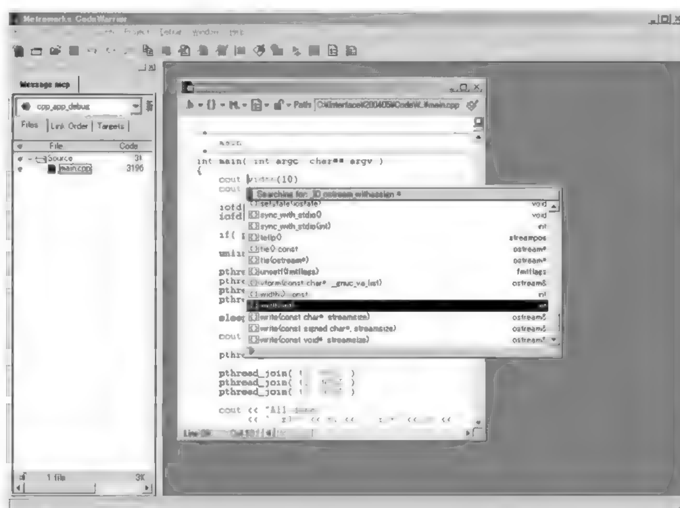

（b）クラス・ブラウザ

図6　CodeWarriorの各種機能


（c）クラス・ブラウザのコード補完機能

● ソース・ブラウザ

CodeWarriorのエディタにはキーワード・ハイライティングなどのほかに，図6（a）のようなソース・ブラウズ機能があり，定義位置の参照などが容易になっています．ほかの統合開発環境でも同様の機能があるのは当たり前のようになっていますが，シンボル定義位置参照，インクルード・ファイルのオープンのほかにマクロ定義位置参照もできるという点では充実していると思います注4．

● クラス・ブラウザ

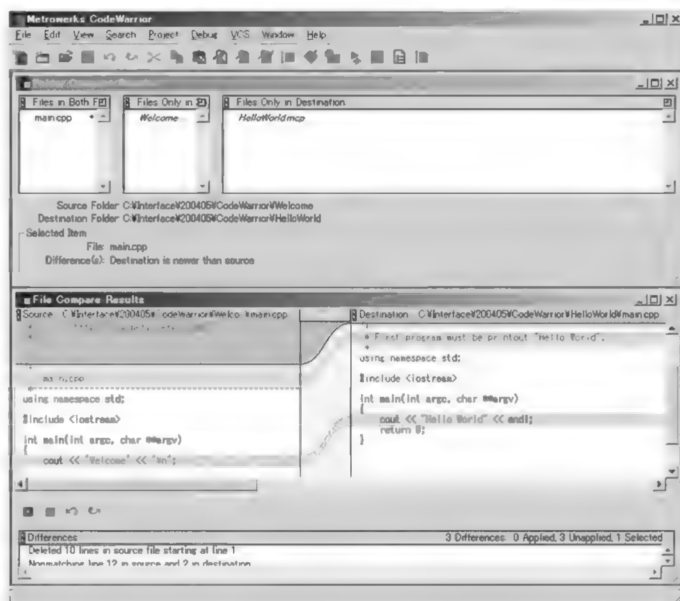
C++で開発を行う場合は図6（b）のクラス・ブラウザを使うと，クラスの継承関係をグラフィカルな表示で確認したり，メソッドの一覧を調べることができます．また，最近の統合開発環境ではおなじみのクラスのメソッドを記述する際に"オブジェクト."または"オブジェクト->"まで記述した時点で図6（c）のようにコードを補完する機能も搭載されています．

● ドッキング

CodeWarrior IDEのVersion5以降ではドッキング・ウィンドウが使えるようになっています．IDE内のウィンドウのタイトル・バーを右クリックするとDocked/Floating/MDI Childの3種類の中からスタイルを選ぶことができます．複数のワーク・スペースを同時に開いて作業する場合には，ドッキングさせると画面内が狭くならずに済むので作業しやすいと思います．ウィンドウをいったんFloatingにしてから移動をすると好みの場所にドッキングすることができるので，テクニックとして知っておくと便利です．

● diff機能

CodeWarriorには図7のようなグラフィカルなdiffツールが


図7　グラフィカルなdiffツール

注4：IDEメニューの Edit→cpp_app_debug Settings ダイアログの Build Extras の設定が"Compiler"か"Language Parser"かにより使える機能内容は変わる．

内蔵されています．
- ファイル間の差分に加えフォルダ間の差分も可能
- 見やすい左右見開き表示
- 差分のマージが可能
- その場で編集も可能

と，このツールは強力で筆者はとても気に入りました．

● バージョン管理ソフトウェアとの連動

ソース・コードをリビジョン管理したい読者もいると思います．専用のツールを使ってもかまいませんが，頻繁に行う操作をIDEから行えば別のツールを立ち上げる手間が省けるので，使ってみるのもよいでしょう．CodeWarriorが連動できるバージョン管理システムは，3種類（CVS，Visual Source Safe，ClearCase）ありますが，ここではCVSとの連動方法を紹介します．

CodeWarriorとCVSと連動するためには，mwCVSというプラグインをインストールする必要があります（米Metrowerks社のサイト　http://www.metrowerks.com/ よりダウンロードすることができる）[注5]．

▶ インストール

ダウンロードしたプラグイン・ファイルを展開したら，ファイルのインストールは手動です．CodeWarriorのインストール・ディレクトリ（通常はC:¥Program Files¥Metrowerks¥CodeWarrior¥）以下に展開したフォルダをコピーします．

▶ mwCVSの設定

CodeWarriorを起動しIDEのメニューからEdit→Version Control Settings...→VCS Setupと選択し，ダイアログのUse costom projectチェックをオンにし，MethodをmwCVSにします．このダイアログ・ボックス内にはLogin Settingsなどの表示欄がありますが，ここでは編集できないのでOKを押してダイアログを閉じます．

次にメニューからVCS→Loginを選択するとダイアログが表示されます．pserverが動作している場合は以下のようになります．

```
CVSROOT    :pserver:ユーザ名@ホスト名:/
                Repository/PathName/cvsroot
Password   あなたのパスワード
```

ローカル・ファイル・システムにリポジトリがある場合はパス名のみでOKです．

Loginに成功するとパスワード・ファイルが"Documents and Settings¥ユーザ名.cvspass"に作られ，VCS Setupダイアログに情報が表示されるようになります．

▶ 使いかた

mwCVSはGNU CVSを元にCodeWarriorのプラグインとして移植されたもので，ほとんどの動作はGNU CVSと同じです[注6]．ただし，mwCVSで扱えるファイルはプロジェクト・ファイル（.mcp）とワーク・スペースに入っているファイルのみで，ワーク・スペースにないファイルやディスク上のフォルダを扱うことはできません．

プロジェクト・ファイルをCVSリポジトリに対して操作する場合は，IDEメニューよりVCS→Project→Update，Commit，Status，Tag，Add，Remove，Logを選択することで行います．

ワーク・スペース内のファイルに対する操作は，ワーク・スペースでファイルを選択した状態でIDEメニューのVCSから操作を選ぶか，または右クリックしてポップアップ・メニューから操作を選択します．

mwCVSを使用してCVSのDifference操作をする場合には，CodeWarrior Diffを使用することができて便利です．

おわりに

CodeWarriorを使用したEmbedded Linuxアプリケーション開発手法の例を挙げて説明しました．今回は組み込み開発ということでARMプロセッサ搭載の評価機を使用しましたが，2種類の評価機ともまったく同じソース・コードで動作確認でき，ターゲットのアーキテクチャを考慮しなければならない部分はほとんどありませんでした．Linux自体が優れたソース・コードのポータビリティを提供していることと，CodeWarrior IDEのGUIがターゲット・アーキテクチャを意識させないデザインと操作性をもっているためだと思います．

組み込み製品に求められるソフトウェアは，年々複雑で高機能化が進んでいるにもかかわらず，Time to Marketの短縮が求められていると思います．コンパイル，リンクはバッチ処理的な作業ですが，デバッグはGUIベースのツールを使用すると作業の効率化を図れることでしょう．

参考文献

(1) 服部貴志；Code Warriorを用いた組み込み向けLinuxプログラミング，Interface，2002年11月号，CQ出版社．
(2) http://www.metrowerks.com/，米Metrowerks社のサイト．
(3) http://www.metrowerks.co.jp/，メトロワークス（株）のサイト．
(4) http://www.cygwin.com/，Cygwinのサイト．
(5) http://www.compaq.co.jp/products/handhelds/pocketpc/old/h3600/spec.html，iPAQ．

ふかせ・しげひろ　深瀬ソフトウェア研究所

注5：Windows版 ftp://ftp.metrowerks.com/pub/updates/VCS/mwcvs_1-3_Win32i_b1015.zipを使用した．
注6：ファイルを削除してコミットするときの動作が通常のCVSの動作と異なっている．

ROM-LinuxChipで実現する

組み込みLinuxのROM化の実例

中島 信行

10年くらい前はOSのROM化といえば，MS-DOSをROM化して，フロッピ・イメージをROMディスクに格納してROM化する方法が取られることが多かったのだが，最近は組み込み分野でもアプリケーションをWindowsやLinuxで作成する場合が多くなった．組み込み系ではハードディスクの使用は，信頼性の点でできれば避けたいため，フラッシュ・メモリなどを使った製品でROM化することが多くなっている．

そこで今回は組み込みの例として(株)ロムウィンのROM-LinuxChip(フラッシュ・メモリ)を使い，LinuxをROM化する例を紹介する．2004年3月号の「特集 Cプログラミングの基礎知識」の続編である．　(筆者)



ROM-Linuxで作る――LinuxのROM化

C君　：先輩，おはようございます．

N先輩：おはよう．

C君　：今度のシステムでWindowsかLinuxでアプリケーションのROM化を検討することになったんですけど，LinuxでROM-LinuxChipを使ってLinuxアプリケーションをROM化するのに，

http://www.rom-win.com/

からフリーのダウンロード版を使ってやってるんですけど難しいんですよね．

N先輩：製品版の開発キットを使えば，難しくないみたいなんだけど．ダウンロード版のROM-Linux Ver.0.94版はLinuxカーネル2.2系までしか対応していないけど，これを使えば，Linuxの理解が深まるから，挑戦してみるのも意味があると思うよ．

C君　：先輩はやったことあるんですか．

N先輩：まだ，途中段階で，ちょっと中断してるんだけどね．

C君　：どこまでいってるんですか．

N先輩：ネットワークの部分がわからないんだ．TCP/IPでの通信はできてるんだけど，sambaやFTPサーバの構築部分がうまくいかないんだ．

C君　：今回はネットワークがいらないので，そこまででよいですから，教えてくださいよ．

N先輩：それじゃ，簡単に説明しようか．まずはパソコンにLinuxのディストリビューションをインストールしてネットワーク(sambaなど)を構築しよう．そうでないと不便だからね．

C君　：そのあたりはできました．ROM-Linux Ver0.94はカーネルが2.2系でないといけないんで，かなり古いですけど，Vine Linux 2.1.5(2.2.18-0vl4.2)をインストールしました．

RTLinuxのインストール

N先輩：それじゃ，RTLinuxは使うのかい．

C君　：それも検討しないといけないんですけど，ROM-Linuxのほうを先に試してるんで，まだ，やってないんですよ．

N先輩：RTLinuxの文献はいくつかあるけど，同じようにしても，Linuxのディストリビューションが違うと，うまくいかないことがあるから，僕が作ったシェルスクリプト(リスト1)で簡単に説明しようか．いくつかのディストリビューションで試してるから，viの-sオプションでスクリプト・ファイルを受け付けるものがインストールされていれば，うまくいくと思うけど．

C君　：vi --helpでみてみると，-s <scriptin>がありますね．

N先輩：余談だけど，LinuxのシェルˑスクリプトをWindowsで編集することがあるかもしれないけど，改行は必ずラインフィードだけにしないとだめだよ．Windowsのようにキャリッジ・リターンがあるとそこで誤動作するんだ．

C君　：僕はviを使ってますから，でも，頭に入れておきます．

N先輩：C君は順応性が高いね．僕は今のところLinuxの使用頻度が少ないから，基本的に手になじんだWindowsのエディタで編集して，ネットワークでLinuxパソコンに転送して動作確認しているんだ．

C君　：そうなんですか．

N先輩：リスト1はRTLinux3.0とRTLinux3.1に対応してるんだけど，今からだったら，RTLinux3.1を使うだろうから．

C君　：はい，その予定です．

N先輩：リスト1はsuコマンドでスーパ・ユーザに移行して使

リスト 1　シェル・スクリプト RTLinuxV3Patch.sh（RT Linux 3.0/3.1版）

```
#!/bin/sh
lsmod >/dev/null 2>&1
if [ $? -ne 0 ]; then export PATH=$PATH:/sbin; fi
RTLMajor=3
if [ "$1" = "i" -o "$1" = "I" ]; then
  RTLMinor=1
  uname -r | grep 2.2.18-rtl>/dev/null
  if [ $? -eq 0 ]; then
    RTLMinor=0
  else
    uname -r | grep 2.4.0-test1-rtl>/dev/null
    if [ $? -eq 0 ]; then RTLMinor=0; fi
  fi
else
  echo "RTLinuxのバージョンを指定してください."
  echo -n "0 : 3.0, 1 : 3.1 ? (デフォルト=1) "
  read RTLMinor
  if [ "${RTLMinor}" != "0" ]; then RTLMinor=1; fi
fi
KernelMajor=2
uname -r | grep 2.2.>/dev/null
if [ $? -eq 0 ]; then
  KernelMinor=2; if [ "${RTLMinor}" = "0" ];
            then KernelRelease=18; else KernelRelease=19; fi
else
  KernelMinor=4; if [ "${RTLMinor}" = "0" ];
            then KernelRelease=0; else KernelRelease=4; fi
fi
RVerP=${RTLMajor}.${RTLMinor}
RVerA=${RTLMajor}${RTLMinor}
KVerP=${KernelMajor}.${KernelMinor}.${KernelRelease}
KVerA=${KernelMajor}${KernelMinor}${KernelRelease}
echo "RTLinux ${RVerP} (Linux ${KVerP}) のインストール中..."
if [ $# -eq 0 ]; then
  TARDIR=/var/tmp
  RTLinuxKernel=rtlinux-${RVerP}.tar.gz
  if [ -r ${TARDIR}/${RTLinuxKernel} ]; then
    RTLinuxKernel=${TARDIR}/${RTLinuxKernel}
  else
    TARDIR=/mnt/cdrom/RTlinuxV3
    if [ -r ${TARDIR}/${RTLinuxKernel} ]; then
      RTLinuxKernel=${TARDIR}/${RTLinuxKernel}
    else
      mount -rt iso9660 /dev/cdrom /mnt/cdrom
      if [ $? -ne 0 ]; then
        echo "/mnt/cdrom がマウントできません."
        exit 1
      fi
      sleep 3
      if [ -r ${TARDIR}/${RTLinuxKernel} ]; then
        RTLinuxKernel=${TARDIR}/${RTLinuxKernel}
      else
        echo "${TARDIR}/${RTLinuxKernel} が見つかりません."
        exit 1
      fi
    fi
  fi
  cd /usr/src
# カーネルの解凍
  KernelTar=
# KernelTar=linux-${KVerP}.tar
  if [ "${RTLMinor}" != "0" ];
                       then KernelTar=linux-${KVerP}.tar; fi
  if [ "${KernelTar}" = "" ]; then
#   プレパッチド・カーネルの解凍
    tar xzf ${TARDIR}/
             rtlinux_kernel_${KernelMajor}_${KernelMinor}.tar.gz
    if [ $? -ne 0 ]; then
      echo "prepatched kernel tar error."
      exit 1
    fi
  else if [ -r ${TARDIR}/${KernelTar}.bz2 ]; then
    if [ -r linux ]; then /bin/rm linux; fi
    tar Ixf ${TARDIR}/${KernelTar}.bz2
    if [ $? -ne 0 ]; then
      echo "${TARDIR}/${KernelTar}.bz2 not bzip2."
      exit 1
    fi
    mv linux linux-${KVerP}-rtl
  else if [ -r ${TARDIR}/${KernelTar}.gz ]; then
    if [ -r linux ]; then /bin/rm linux; fi
    tar xzf ${TARDIR}/${KernelTar}.gz
    if [ $? -ne 0 ]; then
      echo "${TARDIR}/${KernelTar}.gz not tar."
      exit 1
    fi
    mv linux linux-${KVerP}-rtl
  else
    echo "${TARDIR}/${KernelTar}.bz2 and
${TARDIR}/${KernelTar}.gz not found."
    exit 1
  fi; fi; fi
# RTLinuxを解凍
  tar xzf ${RTLinuxKernel}
  if [ $? -ne 0 ]; then
    echo "RTLinux tar error."
    exit 1
  fi
  if [ -r linux ]; then /bin/rm linux; fi
  if [ "${KernelTar}" != "" ]; then
    ln -sf linux-${KVerP}-rtl linux
    cd linux
    if [ -r ../rtlinux-${RVerP}/kernel_patch-${KVerP} ]; then
      patch -p1 < ../rtlinux-${RVerP}/kernel_patch-${KVerP}
    else if [ -r ../rtlinux-${RVerP}/
           kernel_patch-${KernelMajor}.${KernelMinor} ]; then
      patch -p1 < ../rtlinux-${RVerP}/
                kernel_patch-${KernelMajor}.${KernelMinor}
    else
      echo "kernel_patch not found."
      exit 1
    fi; fi
    cd ..
  else if [ -d rtlinux_kernel_${KVerP}-rtl ]; then
    ln -sf rtlinux_kernel_${KVerP}-rtl linux
  else if [ -d rtlinux_kernel_${KVerP}-test1-rtl ]; then
    ln -sf rtlinux_kernel_${KVerP}-test1-rtl linux
  else
    echo "rtlinux_kernel_${KVerP}-rtl error."
    exit 1
  fi; fi; fi
  cd rtlinux-${RVerP}
  ln -sf /usr/src/linux ./linux
  chmod 666 drivers/rt_com/*.c drivers/rt_com/*.h
fi
if [ $# -eq 0 -o "$1" = "r" -o "$1" = "R" ]; then
  cd /usr/src/linux
# Processor type and features ---> Math emulation                N
#                 MTRR (Memory Type Range Register) support       Y
#                 Symmetric multi-processing support              N
# Loadable module support --->  Enable loadable module support Y
#           Set version information on all symbols for modules Y
#               Kernel module loader                           Y
# General setup --->    Advanced Power Management BIOS support N
# Block devices --->    Loopback device support Y
#           RAM disk support                    Y
# Filesystems --->  DOS FAT fs support          M
#           MSDOS fs support                    M
#           VFAT (Windows-95) fs support        M
#           ISO 9660 CDROM filesystem support   Y
#           Microsoft Joliet CDROM extensions   Y
                                            for ロング・ファイル名
#           NTFS filesystem support (read only) M
  make menuconfig
# make xconfig
  if [ $? -eq 0 ]; then
    make dep clean bzImage modules modules_install
    if [ $? -eq 0 ]; then
      cp arch/i386/boot/bzImage /boot/vmlinuz-${KVerP}-rtl
    else
      echo "make error."
      exit 1
    fi
  fi
fi
if [ $# -eq 0 ]; then
  echo -ne "/default¥r7lDA=rtl${RVerA}¥033/image¥ri">rtl.scr
  echo -ne "image=/boot/vmlinuz-${KVerP}-rtl¥r
           ¥tlabel=rtl${RVerA}¥r¥tread-only¥r
           ¥troot=/dev/hda5¥r¥r¥033ZZ">>rtl.scr
  vi -s rtl.scr /etc/lilo.conf
  if [ $? -ne 0 ]; then vi /etc/lilo.conf; fi
  /bin/rm rtl.scr
fi
```

リスト1　シェル・スクリプトRTLinuxV3Patch.sh(RTLinux 3.0/3.1版)(つづき)

```
if [ $# -eq 0 -o "$1" = "r" -o "$1" = "R" ]; then
  /sbin/lilo
  echo -e "¥aリブートします. rtl${RVerA} を起動します. "
  echo "リブート後に sh $0 i を実行してください. "
  echo -n "リブートしてよろしいですか ? (y/n) "
  read ans
  if [ "${ans}" = "y" -o "${ans}" = "Y" ]; then /sbin/reboot; fi
else if [ "$1" = "i" -o "$1" = "I" ]; then
  cd /usr/src/rtlinux-${RVerP}
  make menuconfig
# make xconfig
  if [ -x /usr/bin/kgcc ]; then
#   RedHat 7.0/7.1, LASER5 7.1の場合, gcc→kgccに変更する
    chmod 644 Makefile
    echo -ne ":.,$s/${CROSS_COMPILE}gcc/kgcc/¥rZZ">rtl.scr
    vi -s rtl.scr Makefile
    if [ $? -ne 0 ]; then vi Makefile; fi
    /bin/rm rtl.scr
  fi
  make dep
  if [ $? -ne 0 ]; then
    echo "make dep error."
    exit 1
  fi
  make
  if [ $? -ne 0 ]; then
    echo "make error."
    exit 1
  fi
  make devices install
  if [ $? -ne 0 ]; then
    echo "make devices install error."
    exit 1
  fi
  cd drivers/rt_com; make install; cd ../..
  sh scripts/insrtl
  /sbin/lsmod
  echo "/usr/rtlinux/bin/rtlinux start"
  echo "or"
  echo "/usr/rtlinux/bin/rtlinux start <programname>"
else if [ "$1" = "h" -o "$1" = "-h" -o "$1" = "--help" ]; then
  echo "使用法1"
  echo "sh $0          RTLinuxカーネル構築 (最初)"
  echo "sh $0 i        RTLinuxインストール (リブート後)"
  echo "使用法2"
  echo "sh $0 r        RTLinuxカーネル再構築 (再調整)"
  echo "sh $0 i        RTLinux再インストール (リブート後)"
fi; fi; fi
exit 0
```

```
Processor type and features ---> Math emulation                          N
                  MTRR (Memory Type Range Register) support              Y
                  Symmetric multi-processing support                     N
Loadable module support --->     Enable loadable module support          Y
             Set version information on all symbols for modules          Y
                  Kernel module loader                                   Y
General setup --->       Advanced Power Management BIOS support          N
Block devices --->       Loopback device support                         Y
                  RAM disk support                                       Y
```

図1　make menuconfigの設定ポイント(RTLinux)

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
append="apm=on"
default=rtl31

image=/boot/vmlinuz-2.2.19-rtl
    label=rtl31
    read-only
    root=/dev/hda5

image=/boot/vmlinuz-2.2.18-0vl4.2
    label=linux
    initrd=/boot/initrd-2.2.18-0vl4.2.img
    read-only
    root=/dev/hda5

other=/dev/hda1
    label=dos
```

図2　lilo.conf(RTLinux)

うんだけど．RTLinuxV3Patch.shを~(ホーム・ディレクトリ)にコピーしてlinux2.2.19.tar.bz2，rtlinux-3.1.tar.gzを/var/tmpにコピーしてから，sh RTLinuxV3Patch.shとすればカーネルが構築されるんだ．linux-2.2.19.tar.bz2，rtlinux-3.1.tar.gzはダウンロードしてるよね．

C君　：rtlinux-3.1.tar.gz(RTLinux3.1)は本誌の2002年8月号の付属CD-ROMに入ってましたし，linux-2.2.19.tar.bz2も別の雑誌に付いてたのをもってます．

N先輩：実行するとlinux-2.2.19.tar.bz2とrtlinux-3.1.tar.gzを解凍して，RTLinux用のパッチをカーネルに当てるんだ．Linuxはカーネル内部の危険領域で排他制御のためにロックされている個所があって，リアルタイム性が落ちているんだけど，RTLinux用のパッチはロックされている部分を極力短くするようにしているんだ．

C君　：最初から，そういうことを考えていれば良かったと思いますけど．

N先輩：そりゃそうだけど．でも，Linux2.6系ではこのあたりがかなり改善されるみたいで，RTLinuxを使わなくても，実用になるアプリケーションが増えるかも知れないね．

C君　：そうなんですか．

N先輩：僕はまだ，試してないから，どの程度，改善されるのかはわからないんだけどね．パッチを当てた後はmake menuconfigでカーネルを再構築するための設定をする画面になるから，ここで図1に示した点に注意して，ほかのところはターゲットに合わせて設定するんだ．

C君　：はい．

N先輩：設定が終わると，

```
make dep clean bzImage modules
                        modules_install
```

でカーネルを再構築して，カーネルが，

```
/boot/vmlinuz-2.2.19-rtl
```

としてコピーされるから，これで起動できるように，

/etc/lilo.confにrt131のラベルを追加しているんだ(図2). root=/dev/hda5はパソコンに合わせて変えないといけないけど. そのあと, メッセージを表示するから, いったん再起動して, さらに, スーパ・ユーザになって, shRTLinuxV3Patch.sh iとするとmake menuconfigのところでRTLinux用の設定を入力するとRTLinuxのモジュールがコンパイルされて, インストールされるんだ.

ROM-LinuxをLinuxで使うために

N先輩：RTLinuxをインストールしたら, 今度は, ROM-Linuxの番だね. ROM-LinuxChipにはフラッシュ・メモリとIBM PC/AT互換機用の拡張BIOSが実装されているんだけど, パソコンのCPU基板のソケットに挿入するんだ. マザーボードにソケットがないと使えないんだけど, そのときは別製品のカード・タイプのものを使うことになるんだ.

C君：とりあえず, そのソケットがあるパソコンで試そうと思っています.

N先輩：ROM-LinuxChipをLinuxで扱うには, デバイス名が必要だけど, チップ全体を表すデバイス名は/dev/romaとなるんだ. このデバイスはカーネル・イメージを格納する/dev/rkaとROMディスクの各ディレクトリやファイルを格納する/dev/rpaとから構成されるんだよ(図3).

C君：なんかわかったようなわからないような.

N先輩：ROM-LinuxChipをこんなふうにROMディスクとして認識させるには, そのためのデバイス・ドライバromlinをカーネルに組み込む必要があるんだけど, そのためのパッチがromlin-0.94.tgzを解凍したromlin-0.94/kernel_patchに,

patch-2.2.10.RomLin0.94
patch-2.2.11.RomLin0.94
patch-2.2.12.RomLin0.94
patch-2.2.13.RomLin0.94
patch-2.2.14.RomLin0.94
patch-2.2.15.RomLin0.94
patch-2.2.16.RomLin0.94
patch-2.2.17.RomLin0.94
patch-2.2.18.RomLin0.94

として収められているから, カーネルのバージョンに合わせてパッチを当てれば良いんだ.

C君：2.2.19用のパッチがないみたいですけど.

N先輩：2.2.18用のパッチで試してみたら, うまくいったから, これを使えば良いよ. こっちもシェル・スクリプト(リスト2)を作ってるから, romlin-0.94Patch.shを~(ホーム・ディレクトリ)にコピーして, romlin-0.94.tgzを/var/tmpにコピーしてから, sh romlin-0.94Patch.shとするとromlin-0.94.tgzを/usr/src以下に解凍するんだ. このときできるromlin-0.94用のパッチをカーネルに当てて, カーネルを再構築するんだ. さっきと同じように, make menuconfigのところで設定が必要になるから, 図4に示した点に注意して設定するんだ.

C君：はい.

N先輩：RAMディスクのサイズはターゲットによっては8192とかでも良いことがあるけど, この辺は一度作成してみれば, どの程度必要かわかるよ.

C君：一度, やってみてから決めます.

N先輩：設定が終わるとmake dep clean bzImage modules modules_installでカーネルを再構築して, カーネルが/boot/vmlinuz-2.2.19-rtl_romとしてコピーされるから, これで起動できるように, /etc/lilo.confにrom2219のラベルを追加しているんだ(図5).

C君：rl2219っていうラベルも追加されてますね.

N先輩：設定を変えて何度か試してみるために追加したんだ. 変な設定にすると起動できなくなる可能性があるから, 最初の状態に戻れるようにしただけなんだ.

C君：そうなんですか.

ROM-LinuxChipの構築

N先輩：最後にROM-LinuxChipの構築だよ. パソコンの環境からターゲットに必要なファイルだけをROM-LinuxChipにコピーするんだけど, これがいちばんたいへんなんだ.

C君：どれが必要か知らないといけないわけですから, 確かにたいへんでしょうね.

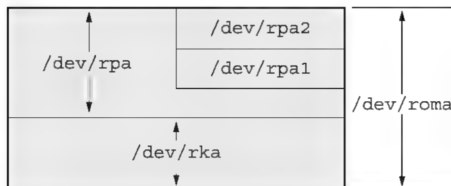


図3 ROM-LinuxChipのROMディスクの構成

```
Block devices ---> Loopback device support          Y
                   RAM disk support                 Y
                          Default RAM disk size 16384
                   RomLinux ROM disk support        Y
                   Initial screen customize support N
```

図4 make menuconfigの設定ポイント(ROM-Linux Ver0.94)

リスト2　シェル・スクリプト romlin-0.94Patch.sh

```
#!/bin/sh
lsmod &>/dev/null
if [ $? -ne 0 ]; then export PATH=$PATH:/sbin; fi
KernelMajor=2
uname -r | grep 2.2.>/dev/null
if [ $? -eq 0 ]; then
  KernelMinor=2
  uname -r | grep 2.2.19>/dev/null
  if [ $? -eq 0 ]; then KernelRelease=19;
                                     else KernelRelease=18; fi
else
  KernelMinor=4
  uname -r | grep 2.4.4>/dev/null
  if [ $? -eq 0 ]; then KernelRelease=4;
                                     else KernelRelease=0; fi
fi
KVerP=${KernelMajor}.${KernelMinor}.${KernelRelease}
KVerA=${KernelMajor}${KernelMinor}${KernelRelease}
if [ $# -eq 0 ]; then
# ROMLIN_TAR_GZ=romlin-0.94.tar.gz
  ROMLIN_TAR_GZ=romlin-0.94.tgz
  TARDIR=/var/tmp
  if [ -r ${TARDIR}/${ROMLIN_TAR_GZ} ]; then
    ROMLIN_TAR_GZ=${TARDIR}/${ROMLIN_TAR_GZ}
  else
    TARDIR=/mnt/cdrom/romlin-0.94
    if [ -r ${TARDIR}/${ROMLIN_TAR_GZ} ]; then
      ROMLIN_TAR_GZ=${TARDIR}/${ROMLIN_TAR_GZ}
    else
      mount -rt iso9660 /dev/cdrom /mnt/cdrom
      if [ $? -ne 0 ]; then
        echo /mnt/cdrom がマウントできません.
        exit 1
      fi
      sleep 3
      if [ -r ${TARDIR}/${ROMLIN_TAR_GZ} ]; then
        ROMLIN_TAR_GZ=${TARDIR}/${ROMLIN_TAR_GZ}
      else
        echo ${TARDIR}/${ROMLIN_TAR_GZ} が見つかりません.
        exit 1
      fi
    fi
  fi
# ROM-Linux パッケージを /usr/src 以下に解凍
  cd /usr/src
  if [ -r ${ROMLIN_TAR_GZ} ]; then
    tar zxvf ${ROMLIN_TAR_GZ}
  else
    echo ${ROMLIN_TAR_GZ} not found.
    exit 1
  fi
# カーネルにパッチを当てる
  cd linux
  RomPatchVer=${KVerP}
  if ! [ -r ../romlin-0.94/kernel_patch/patch-${RomPatchVer}.
                      RomLin0.94 ]; then RomPatchVer=2.2.18;
fi
  patch -p1 < ../romlin-0.94/kernel_patch/patch-${RomPatchVer}.

RomLin0.94
  if [ $? -ne 0 ]; then
    echo patch error.
    exit 1
  fi
fi
if [ $# -eq 0 -o "$1" = "r" -o "$1" = "R" ]; then
  cd /usr/src/linux
# カーネルを再構築
# Block devices ---> Loopback device support    Y
#             RAM disk support                  Y
#             Default RAM disk size         16384
#             RomLinux ROM disk support         Y
#             Initial screen customize support  N
  make menuconfig
# make xconfig
  if [ $? -eq 0 ]; then
    if [ $# -eq 0 ]; then
      make dep clean bzImage modules modules_install
      if [ $? -eq 0 ]; then
        cp arch/i386/boot/bzImage /boot/vmlinuz-`uname -r`_rom
        cp arch/i386/boot/bzImage /boot/vmlinuz-${KVerP}-rom
      else
```

```
        echo make error.
        exit 1
      fi
    else
      make dep clean bzImage modules modules_install
#     make dep clean zImage modules modules_install
      if [ $? -eq 0 ]; then
        cp arch/i386/boot/bzImage /boot/vmlinuz-${KVerP}-rom
#       cp arch/i386/boot/zImage /boot/vmlinuz-${KVerP}-rom
      else
        echo make error.
        exit 1
      fi
    fi
  fi
fi
if [ $# -eq 0 ]; then
# カーネルを lilo やその他のブートローダで起動できるように設定する
# この時, カーネルにコマンド・ライン・パラメータとして romlin_rw=1 を渡すよう
# にします. これによってシステム開発時には ROM-LinuxChip を書き込み可能な
# ブロック・デバイスとして認識させます.
# 例えば lilo の場合は各イメージの設定の中に
#   append = "romlin_rw=1"
# を追加します.
# loadlin の場合はコマンド・ライン・パラメータにそのまま romlin_rw=1 を追加し
# ます. その他の設定については lilo や, 他のローダのマニュアルを参照して下
# さい.
  echo -ne "/image¥r0image=/boot/vmlinuz-`uname -
                                               r`_rom¥r">rom.scr
  echo -ne "¥tappend = ¥"romlin_rw=1¥"¥r¥tlabel=rom${KVerA}
                 ¥r¥tread-only¥r¥troot=/dev/hda5¥r">>rom.scr
  echo -ne "¥rimage=/boot/vmlinuz-${KVerP}-rom¥r">>rom.scr
  echo -ne "¥tappend = ¥"romlin_rw=1¥"¥r¥tlabel=rl${KVerA}¥r
              ¥tread-only¥r¥troot=/dev/hda5¥r¥033ZZ">>rom.scr
  vi -s rom.scr /etc/lilo.conf
  if [ $? -ne 0 ]; then vi /etc/lilo.conf; fi
  /bin/rm rom.scr
# もし pam を使用していないシステムの場合は utils/Makefile の PFLAGS と
                         PLIBS をコメント・アウトしてから make して下さい.
  echo -ne "/PFLAGS =¥ri#¥033/PLIBS =¥ri#¥033ZZ">rom.scr
  vi -s rom.scr /usr/src/romlin-0.94/utils/Makefile
  if [ $? -ne 0 ]; then vi /usr/src/romlin-0.94/utils/
                                                  Makefile; fi
  /bin/rm rom.scr
  if ! [ -d /rom ]; then mkdir /rom; fi
# ROM-LinuxChipマウント
  echo -ne ":$¥romount /dev/rpa1 /rom¥rmount /dev/rpa2 /rom/
                                       usr¥r¥033ZZ">rom.scr
  vi -s rom.scr /etc/rc.d/rc.local
  if [ $? -ne 0 ]; then vi /etc/rc.d/rc.local; fi
  /bin/rm rom.scr
fi
if [ $# -eq 0 -o "$1" = "r" -o "$1" = "R" ]; then
  /sbin/lilo -D rom${KVerA}
# ROM-Linux ユーティリティをインストールします.
  cd /usr/src/romlin-0.94
  make
  make install
  if [ $# -eq 0 -a $? -eq 0 ]; then
    echo -ne "¥aリブートします. rom${KVerA} を起動します. "
    echo "リブート後に sh ROM-LinuxChip.sh を実行してください. "
    echo -n "リブートしてよろしいですか ? (y/n) "
    read ans
    if [ "${ans}" = "y" -o "${ans}" = "Y" ]; then /sbin/reboot;
fi
  fi
else if [ "$1" = "h" -o "$1" = "-h" -o "$1" = "--help" ]; then
  echo 使用法1
  echo "sh $0          ROM-Linuxカーネル構築 (最初)"
  echo 使用法2
  echo "sh $0 r        ROM-Linuxカーネル再構築 (再調整)"
fi; fi
exit 0
```

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
append="apm=on"
default=rtl31

image=/boot/vmlinuz-2.2.19-rtl_rom
    append = "romlin_rw=1"
    label=rom2219
    read-only
    root=/dev/hda5

image=/boot/vmlinuz-2.2.19-rom
    append = "romlin_rw=1"
    label=rl2219
    read-only
    root=/dev/hda5

image=/boot/vmlinuz-2.2.19-rtl
    label=rtl31
    read-only
    root=/dev/hda5

image=/boot/vmlinuz-2.2.18-0vl4.2
    label=linux
    initrd=/boot/initrd-2.2.18-0vl4.2.img
    read-only
    root=/dev/hda5

other=/dev/hda1
    label=dos
```

図5　lilo.conf(ROM-Linux)

```
[root@ROMLIN romlin]# ldd /bin/sh
    libtermcap.so.2 => /lib/libtermcap.so.2 (0x4001d000)
    libc.so.6 => /lib/libc.so.6 (0x40021000)
    /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
[root@ROMLIN romlin]#
```

図6　lddコマンドで共有ライブラリを調べる

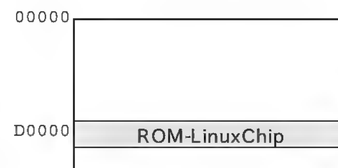


図7　ROM-LinuxChipの拡張BIOSの実装

```
[root@ROMLIN romlin]# df
ファイルシステム      1k-ブロック     使用済      使用可   使用率%  マウント 場所
/dev/hda5          12681660   1453632   10583824      13%  /
/dev/hda3             23333      5353      16776      25%  /boot
/dev/hda8             15863        13      15031       1%  /rom16
/dev/hda6              3963        13       3746       1%  /rom4
/dev/hda7              7931        13       7509       1%  /rom8
/dev/rpa1             12387      7582       4166      65%  /rom
/dev/rpa2              6195      1284       4591      22%  /rom/usr
[root@ROMLIN romlin]#
```

図8　/dev/rpa1, /dev/rpa2でパーティション領域を分ける

N先輩：ある程度，試行錯誤することになると思うけど，本誌の2002年7月号の特集が参考になると思うよ．

C君　：今度，読んどきます．

N先輩：ポイントはlddコマンドで必要な共有ライブラリを調べることかな(図6)．lddコマンドは必要としている共有ライブラリを表示してくれるんだけど，lddコマンドで表示される共有ライブラリが別の共有ライブラリを必要としていることもあるから，結構めんどうなんだけどね．

C君　：確かに，めんどうそうですね．

N先輩：一応，シェル・スクリプト(リスト3)を作ってるから，リスト2のシェル・スクリプト作成されたrl2219でリブート後にsuコマンドでスーパ・ユーザに移行してROM-LinuxChip.shを~(ホーム・ディレクトリ)にコピーしてから，sh ROM-LinuxChip.shとするとネットワークのないROM-LinuxChipが構築できるんだ．rtlapp_module.oがアプリケーションのリアルタイム・モジュールでrtlapp_appが通常のアプリケーションのつもりで書いてるんだけど．

C君　：今回のアプリケーションはこのレベルで良いんですが．

N先輩：ネットワークが必要なときは，sh ROM-LinuxChip.sh nとすればネットワークのあるROM-LinuxChipが構築されるんだけど，こっちは未完成でほかの仕事の関係で中断してるんだ．一応，TCP/IPでの通信はできたんだけど，ホスト名の解決ができない状態になっているから，その先がうまくいかないんだ．シェル・スクリプト(リスト3)の流れを簡単に説明すると，ROM-LinuxChipには拡張BIOSが実装されていて，パソコンの起動時に認識されて実行されるんだ(図7)．そのあと，拡張BIOSはROM-LinuxChipにLinuxカーネルが格納されているかどうかをチェックして，カーネルが格納されていれば，ブート処理を行うんだ．ROM-LinuxChipにカーネルが格納されていないと普通にハードディスクなんかから起動して，リスト2で当てたパッチで組み込まれたデバイス・ドライバromlinでROM-LinuxChipが認識されるとROM-LinuxChip全体は/dev/romaというデバイス名となるんだ．このデバイスを使えるようにするには，添付のr_formatコマンドでフォーマットするんだ．

```
r_format -w -k 10 /dev/roma
```

こうするとカーネル・イメージを格納する/dev/rkaとROMディスクの各ディレクトリやファイルを格納する/dev/rpaが作成されるんだ(-w：ROM-LinuxChipをフォーマットしカーネル領域を確保．-k 10：カーネル領域の大きさの指定)．この状態にするとLinuxの標準のコマンドが使えるようになって，fdiskコマンドで，

```
fdisk /dev/rpa
```

この/dev/rpaをアプリケーションに合わせて，いく

リスト3 シェル・スクリプト ROM-LinuxChip.sh

```
#!/bin/sh
uname -r | grep 2.2.19>/dev/null
if [ $? -eq 0 ]; then KVerP=2.2.19; else KVerP=2.2.18; fi
if [ $# -eq 0 -o "$1" = "n" -o "$1" = "N" ]; then
  APPname=rtlapp
  APPdir=/usr/rtlinux/${APPname}; APPmsg=rtlapp
  if [ -r ${APPdir}/rtlapp_module.o ]; then
    echo "${APPmsg}をROM-LinuxChipに書き込みます."
  else
    echo "${APPdir}/ が見つかりません."
    exit 1
  fi; fi
  if [ -d /rom/usr ]; then umount /rom/usr 2>/dev/null; if [ -d /rom ]; then umount /rom 2>/dev/null; fi; fi
  if ! [ -d /rom ]; then mkdir /rom; fi
  echo -ne "y¥r" | r_format -w -k 10 /dev/roma
# RAM ディスク : 16384
# echo "rpa1 始点 : 1, 終点 : 112 (7MB),    rpa2 始点 : 113, 終点 : 160 (3MB)"
# echo "rpa1 始点 : 1, 終点 : 200 (12.5MB), rpa2 始点 : 201, 終点 : 250 (3.125MB)"
  echo "rpa1 始点 : 1, 終点 : 200 (12.5MB), rpa2 始点 : 201, 終点 : 300 (6.25MB)"
  fdisk /dev/rpa
  mke2fs /dev/rpa1
  mke2fs /dev/rpa2
  mount /dev/rpa1 /rom
  mkdir /rom/usr
  mount /dev/rpa2 /rom/usr
# ROMディスクの構築
  mkdir /rom/bin
  cp -a `which bash; which cat; which cp | grep /cp; which df; which echo; which ls | grep /ls` /rom/bin/
  cp -a `which mount; which ps; which sh` /rom/bin/
# cp -a `which chgrp; which chmod; which chown; which date; which dd; which dmesg; which grep` /rom/bin/
# cp -a `which gzip; which kill; which ln; which login; which mkdir` /rom/bin/
# cp -a `which mknod; which more; which mv | grep /mv; which pwd; which rmdir` /rom/bin/
# cp -a `which sleep; which stty; which su; which sync; which tar; which umount; which uname` /rom/bin/
  mkdir /rom/dev
  cp -a /dev/console /dev/null /dev/tty[0-8] /dev/ttyS[0-3] /dev/ram0 /rom/dev/
  cp -a /dev/roma /dev/rka /dev/rpa /dev/rpa[1-7] /rom/dev/
  cp -a /dev/rtf[0-9] /dev/rtf[1-5][0-9] /dev/rtf6[0-3] /rom/dev/
# cp -a /dev/fd[01] /dev/full /dev/hd[ab] /dev/hda[1-8] /dev/hdb[1-8] /rom/dev/
# cp -a /dev/lp[0-2] /dev/tty /dev/ram /dev/ramdisk /dev/ram[1-7] /rom/dev/
# cp -a /dev/mem /dev/kmem /dev/port /dev/random /dev/urandom /dev/zero /dev/stderr /dev/stdin /dev/stdout /rom/dev/
  mkdir /rom/etc
# cp -a /etc/DIR_COLORS /etc/inittab /rom/etc/
# cp -a /etc/ld.so.cache /etc/ld.so.conf /etc/login.defs /etc/magic /etc/motd /rom/etc/
# cp -a /etc/mtab /etc/profile /etc/shadow /etc/termcap /rom/etc/
# /rom/etc/fstab
  echo "/dev/ram0      /       ext2    defaults    1   1">/rom/etc/fstab
  echo "/dev/rpa2      /usr    ext2    defaults    1   1">>/rom/etc/fstab
  echo "none           /proc   proc    defaults    0   0">>/rom/etc/fstab
  mkdir /rom/etc/rc.d
# cp -a /etc/rc.d/rc.sysinit /etc/rc.d/rc /rom/etc/rc.d/
# cp -a /etc/rc.d/rc.0 /etc/rc.d/rc.6 /etc/rc.d/rc.M /etc/rc.d/rc.S /etc/rc.d/rc.local /rom/etc/rc.d/
  mkdir /rom/lib
  cp -a /lib/ld-2.1.3.so /lib/ld-linux.so.2 /lib/libc-2.1.3.so /lib/libc.so.6 /rom/lib/
  cp -a /lib/libproc.so.2* /lib/libtermcap.so.2* /rom/lib/
  mkdir /rom/root
  mkdir /rom/proc
  mkdir /rom/var
  mkdir /rom/var/log
# cp -a /var/log/messages /var/log/wtmp /rom/var/log/
  mkdir /rom/var/run
# cp -a /var/run/utmp /rom/var/run/
  mkdir /rom/var/spool
  mkdir /rom/var/tmp
  mkdir /rom/sbin
# /sbin/agetty
# cp -a /sbin/clock /sbin/e2fsck /sbin/fsck /sbin/fsck.ext2 /rom/sbin/
# /sbin/init
  cp -a /sbin/insmod /sbin/lsmod /rom/sbin/
# cp -a /sbin/halt /sbin/ldconfig /sbin/mingetty /sbin/modinfo /sbin/modprobe /rom/sbin/
# cp -a /sbin/update /sbin/reboot /sbin/shutdown /rom/sbin/
  if [ 0 -eq 1 ]; then
#   /sbin/initから起動 (未完成)
    cp -a `which chgrp; which chmod; which date; which dmesg; which domainname; which ln` /rom/bin/
    cp -a `which sleep; which true; which umount; which uname` /rom/bin/
    cp -a /sbin/clock /sbin/e2fsck /sbin/fsck /sbin/fsck.ext2 /sbin/getkey /sbin/halt /sbin/hwclock /sbin/init /rom/sbin/
    cp -a /sbin/mingetty /sbin/reboot /sbin/shutdown /sbin/sulogin /sbin/swapon /sbin/update /rom/sbin/
    cp -a /dev/initctl /rom/dev/
    cp -a /etc/adjtime /etc/inittab /rom/etc/
    cp -a /etc/rc.d/rc.sysinit /etc/rc.d/rc /etc/rc.d/rc.local /rom/etc/rc.d/
    mkdir /rom/etc/rc.d/rc3.d/
    cp -a /etc/rc.d/rc3.d/* /rom/etc/rc.d/rc3.d/
    cp -a /var/run/utmp /rom/var/run/
```

リスト3　シェル・スクリプトROM-LinuxChip.sh（つづき）

```
    cp -a /var/log/wtmp /rom/var/log/
  else
#   /rom/etc/init
    echo "#!/bin/sh">/rom/etc/init
    echo "/bin/mount -av">>/rom/etc/init
    echo "for f in /usr/local/etc/*; do if [ -f ¥$f ]; then cp -a ¥$f /home/${APPname}/; fi; done">>/rom/etc/init
    echo "/sbin/insmod /home/${APPname}/rtl.o">>/rom/etc/init
    echo "/sbin/insmod /home/${APPname}/rtl_time.o">>/rom/etc/init
    echo "/sbin/insmod /home/${APPname}/rtl_sched.o">>/rom/etc/init
    echo "/sbin/insmod /home/${APPname}/rtl_posixio.o">>/rom/etc/init
    echo "/sbin/insmod /home/${APPname}/rtl_fifo.o">>/rom/etc/init
    echo "/sbin/insmod /home/${APPname}/rt_com.o">>/rom/etc/init
    echo "/sbin/insmod /home/${APPname}/rtlapp_module.o">>/rom/etc/init
#   キーマップのロード
    echo "# Load keymap">>/rom/etc/init
    echo "KEYMAP=">>/rom/etc/init
    echo "if [ -f /etc/sysconfig/keyboard ]; then . /etc/sysconfig/keyboard; fi">>/rom/etc/init
    echo "if [ -n ¥"¥$KEYTABLE¥" -a -d ¥"/usr/lib/kbd/keymaps¥" ]; then KEYMAP=¥$KEYTABLE; fi">>/rom/etc/init
    echo "# Since this takes in/output from stdin/out, we can't use initlog">>/rom/etc/init
    echo "if [ -n ¥"¥$KEYMAP¥" ]; then loadkeys ¥$KEYMAP </dev/tty0 >/dev/tty0; fi">>/rom/etc/init
#   日本語表示
    echo "[ -f /etc/sysconfig/i18n ] && . /etc/sysconfig/i18n">>/rom/etc/init
    echo "[ ¥"¥$LANG¥" != ¥"¥" ] && export LANG">>/rom/etc/init
    echo "[ ¥"¥$LANGUAGE¥" != ¥"¥" ] && export LANGUAGE">>/rom/etc/init
    echo "[ ¥"¥$LC_ALL¥" != ¥"¥" ] && export LC_ALL">>/rom/etc/init
    echo "localedef -f eucJP -i ja_JP ja_JP.eucJP">>/rom/etc/init
    echo "echo ¥"漢字表示テスト¥"">>/rom/etc/init
    if [ "$1" = "n" -o "$1" = "N" ]; then
#     ネットワーク
      echo "if [ -f /etc/sysconfig/network ]; then . /etc/sysconfig/network; fi">>/rom/etc/init
#     echo "# Set the hostname.">>/rom/etc/init
#     echo "hostname ¥$HOSTNAME">>/rom/etc/init
#     echo "echo ¥$HOSTNAME >/etc/HOSTNAME">>/rom/etc/init
      echo "[ ¥"¥$HOSTNAME¥" != ¥"¥" ] && export HOSTNAME">>/rom/etc/init
      echo "/etc/rc.d/init.d/network start">>/rom/etc/init
#     echo "/etc/rc.d/init.d/portmap start">>/rom/etc/init
#     echo "/etc/rc.d/init.d/identd start">>/rom/etc/init
      echo "/etc/rc.d/init.d/inet start">>/rom/etc/init
      echo "/etc/rc.d/init.d/smb start">>/rom/etc/init
    fi
    echo "/home/${APPname}/rtlapp_app &">>/rom/etc/init
    echo "/bin/bash">>/rom/etc/init
    chmod +x /rom/etc/init
  fi
  mkdir /rom/usr/bin
  cp -a `which du; which free; which less` /rom/usr/bin/
# cp -a `which chfn; which chsh; which cut; which diff; which dircolors` /rom/usr/bin/
# cp -a `which file; which find; which groff; which head; which lesskey; which lesspipe.sh` /rom/usr/bin/
# cp -a `which passwd; which setterm; which tee; which tty; which which | grep /which` /rom/usr/bin/
# cp -a `which elvis; which elvprsv; which elvrec; which vi; which view` /rom/usr/bin/
  mkdir /rom/usr/sbin
# cp -a /usr/sbin/rdev /usr/sbin/adduser /usr/sbin/useradd /usr/sbin/userdel /usr/sbin/usermod /rom/usr/sbin/
  mkdir /rom/usr/lib
  cp -a /usr/lib/libncurses.so.4* /rom/usr/lib/
  mkdir /rom/usr/local /rom/usr/local/etc
  mkdir /rom/home
  mkdir /rom/mnt
  mkdir /rom/tmp
  ldconfig -v -r /rom/
# キーマップのロード
  mkdir /rom/etc/sysconfig
  cp -a /etc/sysconfig/keyboard /rom/etc/sysconfig/
  mkdir /rom/usr/lib/kbd
  mkdir /rom/usr/lib/kbd/keymaps
  mkdir /rom/usr/lib/kbd/keymaps/i386
  mkdir /rom/usr/lib/kbd/keymaps/i386/qwerty
  cp -a /usr/lib/kbd/keymaps/i386/qwerty/jp106* /rom/usr/lib/kbd/keymaps/i386/qwerty/
  cp -a `which gunzip` /rom/bin/
  cp -a `which loadkeys` /rom/usr/bin/
# 日本語表示 （未完成）
  cp -a `which env; which kon; which localedef` /rom/usr/bin/
  cp -a /etc/kon.cfg /rom/etc/
  cp -a /etc/sysconfig/i18n /rom/etc/sysconfig/
  mkdir /rom/usr/share
  mkdir /rom/usr/share/i18n
  mkdir /rom/usr/share/i18n/locales
  cp -a /usr/share/i18n/locales/ja_JP* /rom/usr/share/i18n/locales/
  mkdir /rom/usr/share/i18n/charmaps
  cp -a /usr/share/i18n/charmaps/EUC-JP /rom/usr/share/i18n/charmaps/
  mkdir /rom/usr/share/locale
  cp -a /usr/share/locale/ja_JP.* /rom/usr/share/locale/
# 日本語入力 （未完成）
```

リスト3 シェル・スクリプト ROM-LinuxChip.sh(つづき)

```
  if [ "$1" = "n" -o "$1" = "N" ]; then
#   ネットワーク環境の構築 (TCP/IP)
    cp -a `which awk; which basename; which egrep; which gawk; which grep; which hostname; which ipcalc` /rom/bin/
    cp -a `which nice; which ping; which rm | grep /rm; which sed; which touch` /rom/bin/
    cp -a `which export` /rom/usr/bin/
    cp -a /etc/host.conf /etc/hosts /etc/hosts.allow /etc/hosts.deny /etc/inetd.conf /etc/protocols /rom/etc/
    cp -a /etc/resolv.conf /etc/services /etc/sysctl.conf /rom/etc/
    mkdir /rom/etc/rc.d/init.d
    cp -a /etc/rc.d/init.d/functions /etc/rc.d/init.d/network /etc/rc.d/init.d/inet /rom/etc/rc.d/init.d/
    cp -a /etc/sysconfig/init /etc/sysconfig/network /rom/etc/sysconfig/
    mkdir /rom/etc/sysconfig/network-scripts
    cp -a /etc/sysconfig/network-scripts/ifcfg-eth0 /rom/etc/sysconfig/network-scripts/
    cp -a /etc/sysconfig/network-scripts/ifcfg-lo /rom/etc/sysconfig/network-scripts/
    cp -a /etc/sysconfig/network-scripts/ifup /etc/sysconfig/network-scripts/ifup-post /rom/etc/sysconfig/network-scripts/
    cp -a /etc/sysconfig/network-scripts/ifup-aliases /rom/etc/sysconfig/network-scripts/
    cp -a /etc/sysconfig/network-scripts/ifup-routes /rom/etc/sysconfig/network-scripts/
    cp -a /etc/sysconfig/network-scripts/network-functions /rom/etc/sysconfig/network-scripts/
    cp -a /usr/sbin/inetd /usr/sbin/tcpd /rom/usr/sbin/
    cp -a /sbin/ifconfig /sbin/ifup /sbin/initlog /sbin/killall5 /sbin/pidof /sbin/route /sbin/sysctl /rom/sbin/
    cp -a /lib/libcrypt* /lib/libdl-2.1.3.so /lib/libdl.so.2 /lib/libm-2.1.3.so /lib/libm.so.6 /lib/libnsl* /rom/lib/
    cp -a /lib/libresolv* /lib/libpam.so.0* /rom/lib/
    cp -a /usr/lib/libreadline.so* /rom/usr/lib/
    mkdir /rom/var/lock
    mkdir /rom/var/lock/subsys
    cp -a /var/lock/subsys/network /var/lock/subsys/inet /rom/var/lock/subsys
    cp -a /var/run/netreport/ /rom/var/run/
#   FTP (未完成)
    cp -a /usr/sbin/proftpd /rom/usr/sbin/
    cp -a /etc/group /etc/passwd /etc/proftpd.conf /etc/shutmsg /rom/etc/
    cp -a /etc/rc.d/init.d/proftpd /rom/etc/rc.d/init.d/
#   Samba (未完成)
    cp -a `which smbmount` /rom/usr/bin/
    cp -a /etc/lmhosts /etc/printcap /etc/smb.conf /etc/smbpasswd /etc/smbusers /etc/MACHINE.SID /rom/etc/
    cp -a /etc/rc.d/init.d/smb /rom/etc/rc.d/init.d/
    cp -a /usr/sbin/nmbd /usr/sbin/smbd /usr/sbin/swat /rom/usr/sbin/
    cp -a /sbin/mount.smbfs /rom/sbin/
    mkdir /rom/var/log/samba
    mkdir /rom/var/spool/samba
    mkdir /rom/etc/codepages
    cp -a /etc/codepages/codepage.932 /etc/codepages/unicode_map.850 /etc/codepages/unicode_map.932 /rom/etc/codepages/
    mkdir /rom/etc/pam.d
    cp -a /etc/pam.d/samba /rom/etc/pam.d/
    mkdir /rom/etc/logrotate.d
    cp -a /etc/logrotate.d/samba /rom/etc/logrotate.d/
  fi
  pushd /lib
  ls -l `for i in /rom/bin/* /rom/sbin/* /rom/usr/bin/*; do ldd $i | awk '{print $1 "¥n" $3}'; done | sort | uniq`
  popd
#
  # syslogd, klogd 削除 (ログが溜まると困るため)
  for delnam in /rom/usr/sbin/syslogd /rom/usr/sbin/klogd; do if [ -r ${delnam} ]; then /bin/rm ${delnam}; fi; done
#
  umount /rom/usr
  # romlin-0.94カーネル再構築
  if [ "$1" = "r" -o "$1" = "R" -o "$2" = "r" -o "$2" = "R" ]; then sh romlin-0.94Patch.sh r; fi
  # /etc/r_loader.conf変更(kernel=/boot/vmlinuz-${KVerP}-rom)
  echo -ne "/ROM-LinuxChip¥r/default¥rDAdefault = 0¥033/vmlinuz¥rDAvmlinuz-${KVerP}-rom¥033ZZ">chip.scr
  vi -s chip.scr /etc/r_loader.conf
  if [ $? -ne 0 ]; then vi /etc/r_loader.conf; fi
  /bin/rm chip.scr
  if ! [ -d /rom/home/${APPname} ]; then mkdir /rom/home/${APPname}; fi
  cp /lib/modules/`uname -r`/misc/* /rom/home/${APPname}
  cp ${APPdir}/rtlapp_module.o /rom/home/${APPname}
  cp ${APPdir}/rtlapp_app /rom/home/${APPname}
  umount /rom
  r_loader -w
  if [ $? -ne 0 ]; then echo "ROM-LinuxChipの構築に失敗しました."; exit 1; fi
  r_loader -mk "rw"
  if [ $? -ne 0 ]; then echo "ROM-LinuxChipの構築に失敗しました."; exit 1; fi
  r_loader -mk "romlin_rw=1"
  if [ $? -ne 0 ]; then echo "ROM-LinuxChipの構築に失敗しました."; exit 1; fi
  r_loader -r
  mount /dev/rpa1 /rom
  mount /dev/rpa2 /rom/usr
  df
  echo "ROM-LinuxChipを構築しました."
else if [ "$1" = "h" -o "$1" = "-h" -o "$1" = "--help" ]; then
  echo -e "使用法1¥nsh $0          ROM-LinuxChip構築 (ネットワークなし)"
  echo -e "使用法1¥nsh $0   r      ROM-LinuxChip構築 (ネットワークなし : カーネル再構築)"
  echo -e "使用法2¥nsh $0 n        ROM-LinuxChip構築 (ネットワーク付き)"
  echo -e "使用法2¥nsh $0 n r      ROM-LinuxChip構築 (ネットワーク付き : カーネル再構築)"
fi; fi
exit 0
```

つかのパーティション領域に分ければ良いんだ(図8).
これらのパーティション領域にmke2fsコマンドで,

```
mke2fs /dev/rpa1
mke2fs /dev/rpa2
```

のようにLinux 2ndファイル・システム(ext2fs)を作成して,

```
mkdir /rom
mkdir /rom/usr
mount /dev/rpa1 /rom
mount /dev/rpa2 /rom/usr
```

のようにマウントしてから,必要なファイルをコピーしていくことになるんだ.

C君　:なんか,めんどうそうですね.Linuxディストリビューションから直接インストールして,いらないファイルを削除するといったことができれば良いんですけどね.

N先輩:フラッシュ・メモリの容量が少ないから,Linuxディストリビューションから直接インストールはできないんだ.

C君　:残念ですね.

N先輩:製品版だと自動コンパクト化ツールがあって,楽ができるみたいだけどね.カーネルの書き込みは添付のr_loaderコマンドで行うんだけど,このコマンドは/etc/r_loader.conf(図9)を参照するから,これもアプリケーションに合わせて書き換えているんだ.[boot]セクションではブート・パラメータを指定するんだけど,delayで起動時にキー入力を待つ時間を秒数で指定して,defaultでROM-LinuxChipから起動するか(default=0),ディスクから起動するか(default=1)指定するんだ.[kernel]セクションではカーネルに渡すパラメータを指定しているんだ.図9はroot = /dev/ram0でルート・デバイスを/dev/ram0としているだけだけど,

romlin_rw … 書き込み可能にするかどうかの指定

ro … ルート・デバイスを読み込み専用でマウント

rw … ルート・デバイスを読み書き可能でマウント

なんかを指定できるんだ.最後の[image]セクションではkernelパラメータでカーネル・イメージのファイルを指定しているんだ.リスト3はr_loaderコマンドのオプション・スイッチでカーネル・パラメータをrwとromlin_rw=1と修正して読み書き可能としているんだけど.

```
r_loader -mk "rw"
r_loader -mk "romlin_rw=1"
```

必要なファイルのコピーはcpコマンドで-aオプションを付けて行う必要があって,-aオプションを付けると,できる限り,元のファイルの構成と属性を保持してコピーされるんだ.どのファイルをコピーするかが大事なんだけど.

C君　:リスト3にも試行錯誤の痕跡がありますね.

N先輩:そうなんだ.ディストリビューションやアプリケーションによっても,必要なファイルは違ってくるから,ある程度の試行錯誤は必要だと思うよ.

C君　:そうでしょうね.

N先輩:今回のリスト3は/dev/rpa1を/(ルート)にマウント

```
# ROM-Linux loader configration file
#
#
# boot section parameters
#   delay = n       wait time ( n sec ) at system boot
#   default = n     n ; 0 boot from ROM-LinuxChip
#                       1 boot from normal disk
[boot]
  delay = 5
  default = 0
#
#
# kernel section parameters
#   any parameters for kernal are available
#   root = xxx  or  ramdisk_size = xxx  or  rw   .... etc.
#   for ROM-LinuxChip following parameters are available
#   romlin_rw = n         n ; 0  device is read only
#                             1  device is writable
#                         default value is 0
#   romlin_image = n      n is minor number of /dev/rpa
                                                    that
#                         will be loaded by load ramdisk
                                                function
#                         default value is 1
#   romlin_initrd = ofst  offset initrd started
[kernel]
  root = /dev/ram0
  # romlin_rw = 1
#
#
# image section parameters
#   kernel = (full path)
#   initrd = (full path)
[image]
  kernel = /boot/vmlinuz-2.2.19-rom
```

図9　r_loader.confでカーネルの書き込みをする

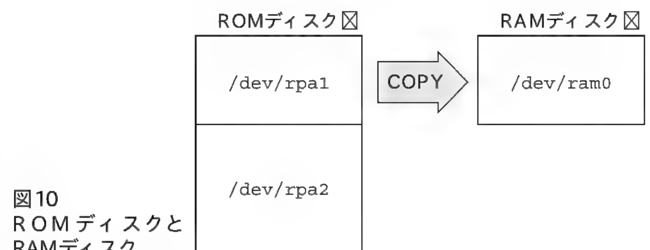


図10　ROMディスクとRAMディスク

```
/dev/ram0     /       ext2    defaults   1   1
/dev/rpa2     /usr    ext2    defaults   1   1
none          /proc   proc    defaults   0   0
```

図11　/etc/fstabでdefaultsとして読み書き可能とする

して，/dev/rpa2を/usrにマウントしているんだけど，/usr以下(/dev/rpa2)をROMディスクとしていて，/dev/rpa1は起動時にRAMディスクにコピーされてマウントされるんだ(図10)．それに，swapを通常使わない設定にしてるから，メモリは十分に積んでおかないといけないんだ．/tmp，/var，/etcなんかは書き込みがされることがあるから，RAMディスクは当然，読み書き可能だけど，ROMディスクのほうも，/usr以下はシステムで書き込まれることはないから，図11の/etc/fstabでdefaultsとして読み書き可能にしているんだ．defaultsだとブート時に自動的に標準でマウントされるから，読み書き可能になるんだけど(ちなみにdefaultsをroに変えると読み込み専用でマウントされる)．

C君　：どうして，そんなふうにしているんですか．

N先輩：プログラムを書き換えるときのためだよ．

C君　：なるほどね．

N先輩：キーボードとCRTがあるシステムだったら，起動時にカーネルに渡すコマンド・ライン・パラメータを指定できるから，/dev/rpa2を読み込み専用にして，プログラムを更新するときだけ，コマンド・ライン・パラメータでromlin_rw = 1を指定して一時的に読み書き可能に変えるといったアプローチもあると思うけどね．ターゲットの動作時に読み込み専用にしてあれば，何らかの誤動作で書き換えられる危険性もなくなるから，より安全なんだ．

C君　：そうですね．

N先輩：ネットワークが不要なシステムでもネットワークでつながれば，プログラムのバージョンアップが楽なんだけど，今のところできてないから，TCP/IPでプログラムを転送するユーティリティを自作することを考えているんだけどね．

C君　：やっぱり，FTPかsambaでつながってたほうが便利そうですね．ネットワークに対応したやつを早く作ってくださいよ．

N先輩：そのうち，時間が取れたときに再挑戦してみるつもりだよ．

おわりに

今回は組み込みの実例としてROM-LinuxChipを紹介しました．ROM化の例としては特殊な例ですが，LinuxのROM化を検討する際の参考にしてください．

参考文献

(1) A.コーニング著，中村明訳；Cプログラミングの落し穴，(株)トッパン，1990年10月30日初版第3刷
(2) B.W.カーニハン，D.M.リッチー著，石田晴久訳；プログラミング言語C第2版，共立出版(株)，1989年6月15日初版1刷
(3) 平林雅英；ANSI C言語辞典，技術評論社，平成元年10月25日初版第1刷
(4) John H.Crawford+Patrick P.Gelsinger，岩谷宏訳；80386プログラミング，(株)工学社，昭和63年7月25日初版
(5) インテルジャパン(株)，i486マイクロプロセッサ プログラマーズ・リファレンス・マニュアル，1991年12月25日初版第1刷
(6) TECH I Vol.5，技術者のためのUnix系OS入門，CQ出版(株)，2000年7月1日
(7) 森友一朗，薬師輝久，馬場秀忠；RTLinuxリアルタイム処理プログラミングハンドブック，(株)秀和システム，2000年12月8日初版第1刷
(8) Michael Beck，Harald Bohme，Mirko Dziadzka，Ulrich Kunitz，Robert Magnus，Dirk Verworner著；(株)クイック訳，Linuxカーネルインターナル，(株)アスキー，1999年9月21日初版第2刷
(9) 奥脇学；Linuxプログラミングガイド，(株)秀和システム，2000年10月20日初版第1刷
(10) 船木陸議；RTLinux2.2系の導入と詳細，インターフェース，2001年6月号
(11) 山本繁寿/有末一寿；組み込み機器とプログラミング言語，インターフェース，2002年3月号
(12) 山本繁寿/有末一寿；組み込み機器とデバッグ環境，インターフェース，2002年3月号
(13) 西田 亙；オリジナルルートファイルシステムの構築，インターフェース，2002年7月号
(14) セサミアン3人組 組込みソフトウェア管理者・技術者育成研究会；長く活躍できる技術者になろう，デザインウェーブマガジン，2003年5月号
(15) セサミアン3人組 組込みソフトウェア管理者・技術者育成研究会；「組み込み」ならではの基礎知識，デザインウェーブマガジン，2003年5月号
(16) TECH I Vol.16，組み込みLinux入門，CQ出版(株)，2003年4月1日
(17) ROM-Linux Install Guide ROM-Linux Ver0.94，(株)ワコムエンジニアリング
(18) ROM-Linux Manual ROM-Linux Ver0.94，(株)ワコムエンジニアリング

なかしま・のぶゆき　(株)Unix

オープンソースのITRON仕様OS

TOPPERSで学ぶRTOS技術

第6回 サービス・コールの概要・その3

岸田 昌巳

今回も引き続き，シミュレーション環境を使いながら各サービス・コールについての説明を行います．

仕様の説明については，前回同様，リターン・パラメータ(返り値)に関して，TOPPERS/JSPで返さない値に関しては括弧付きで表現しています．これらについては，参考のための記述と理解してください．後々の機能拡張などを考えると，返り値として返ってくる来ないに関わらず，エラーチェックは行っておいてください．

表1　同期・通信機能のサービス・コール一覧

セマフォ
ER　sig_sem(ID semid);
ER　isig_sem(ID semid);
ER　wai_sem(ID semid);
ER　pol_sem(ID semid);
ER　twai_sem(ID semid, TMO tmout);
イベント・フラグ
ER　set_flg(ID flgid, FLGPTN setptn);
ER　iset_flg(ID flgid, FLGPTN setptn);
ER　clr_flg(ID flgid, FLGPTN clrptn);
ER　wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
ER　pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
ER　twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout);
データ・キュー
ER　snd_dtq(ID dtqid, VP_INT data);
ER　psnd_dtq(ID dtqid, VP_INT data);
ER　ipsnd_dtq(ID dtqid, VP_INT data);
ER　tsnd_dtq(ID dtqid, VP_INT data, TMO tmout);
ER　fsnd_dtq(ID dtqid, VP_INT data);
ER　ifsnd_dtq(ID dtqid, VP_INT data);
ER　rcv_dtq(ID dtqid, VP_INT *p_data);
ER　prcv_dtq(ID dtqid, VP_INT *p_data);
ER　trcv_dtq(ID dtqid, VP_INT *p_data, TMO tmout);
メール・ボックス
ER　snd_mbx(ID mbxid, T_MSG *pk_msg);
ER　rcv_mbx(ID mbxid, T_MSG **ppk_msg);
ER　prcv_mbx(ID mbxid, T_MSG **ppk_msg);
ER　trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout);

同期・通信機能のサービス・コール

同期・通信機能としては，セマフォ，イベント・フラグ，データ・キュー，メール・ボックスの機能があります．これらは，同期機能と通信機能を別々に提供しているのではなく，ほぼ，どちらの用途にも使える機能が提供されています．

これらのサービス・コールとしては，xxx_sem(xxxはsigやwaiなど)，xxx_flg(同じくxxxにはsetやwaiなど)，xxx_dtq(xxxはsndやrcvなど)，xxx_mbx(xxxはsndやrcvなど)があります．

表1に，これらTOPPERS/JSPでサポートしている機能を掲載します．

状態遷移図から見ると，それぞれの待ちに入るサービス・コールと待ちから解除するサービス・コール以外に，強制的に待ち状態から解除するサービス・コールのみが，タスクのふるまいに関連します(図1，図2)．

セマフォ

● セマフォの概要

セマフォは，排他制御に使用します．排他制御は，同時に利用できないクリティカル・セクションを複数のタスクから共有する場合など，「自分が使用している途中なので，ほかから使用されないようにする」という，ほかを排除したい/使わせたくない場合に使います．

ここで言う管理対象のクリティカル・セクションとは，

- 一度に一つのタスクしかアクセスしてはいけない関数
- あるアドレスのデータ(変数)
- 入出力を行うI/Oなど

といったリソースやコードを指しています．

これらのクリティカル・セクションにアクセスする場合には排他制御が必要になります．たとえば，非同期に動作する複数のタスクが，同じアドレスのメモリを書き換えたい場合などにはセマフォを使用して排他制御を行います(図3)．

通常，クリティカル・セクションは排他制御が必要な小規模

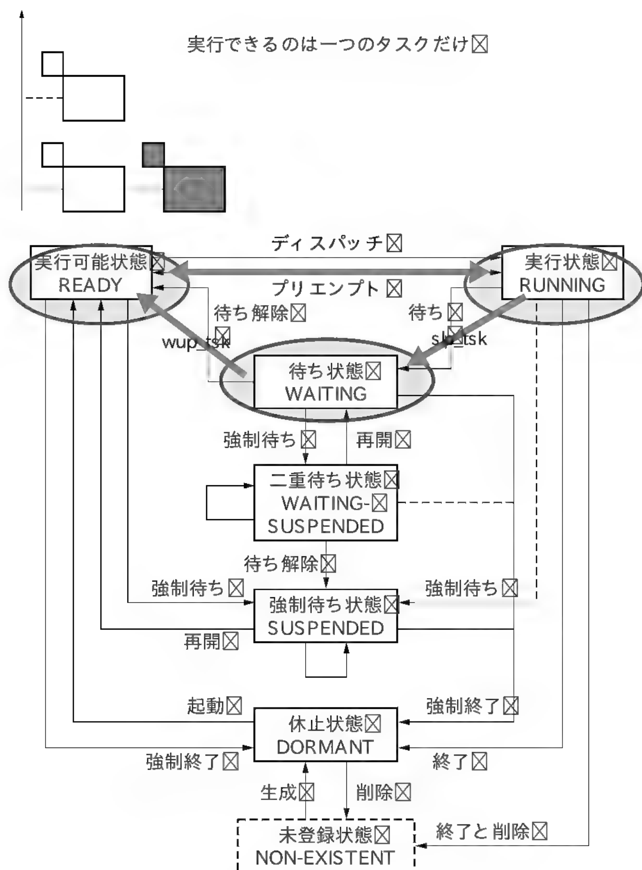


図1　サービス・コールの状態遷移図

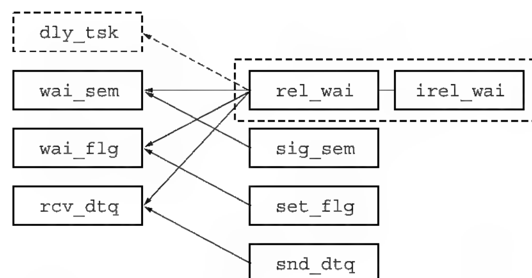


図2　各サービス・コールの関連図



図3　部屋と鍵――排他制御の例

な範囲[注1]を示します(図4)．これ以外には資源をもう少し大きな範囲，同時に使用する複数の資源や，処理の優先度などを元に，グループ単位にまとめた塊をクリティカル・セクションとして扱う場合もあります．

このようなクリティカル・セクションを扱うとき，管理するクリティカル・セクションに対し，セマフォ資源を一つ確保することで，利用可能な状態にあるのか/ないのかを判断することができるようになります．

ここまでは，一つのクリティカル・セクションに対し，一つのセマフォでしたが，排他制御の対象が一つでない場合に利用することもあります．複数個のリソースがあり，このリソースを確保できるかどうかをセマフォで管理する場合です．

たとえるなら，ペンションなどの管理人室に鍵がぶら下がっており，鍵があれば空き室，なければ使用中というものを思い浮かべていただければ良いでしょう(図5)．ここでのセマフォ資源は各部屋の鍵にあたります．セマフォIDが違う場合は，別のペンションを指しているといえます．

この場合，複数の部屋があり，空き部屋か，使用中か，鍵の

注1：もっと大規模な範囲も排他可能だが，大規模な排他制御は性能などの問題を引き起こしやすいため，小さな範囲に収まるよう，設計に考慮しておくことを推奨する．

複数のタスクからfuncAが呼ばれる．
タイミングは任意．

funcSubA1とfuncSubA2はこの順序で実行したいが，ほかのタスクとの関係から，＊の位置でタスク・スイッチすることもあるとする．
このとき(＊の位置でタスク・スイッチしたとき)，funcAがスイッチした後のタスクから呼ばれるとfuncSubA1，funcSubA2の順序で実行できない．
funcSubA1，funcSubA1，funcSubA2，funcSubA2の順序になってしまう．

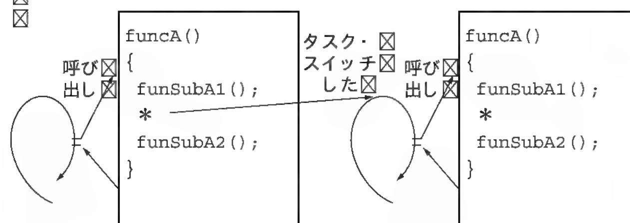


図4　クリティカル・セクション

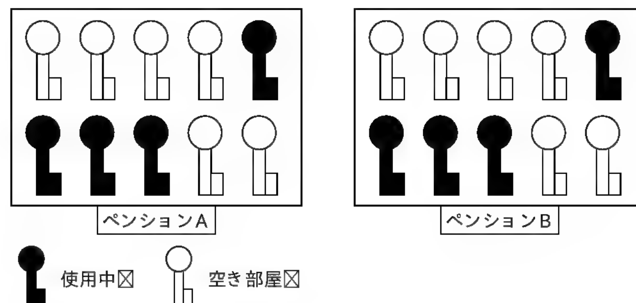


図5　セマフォの概念図1

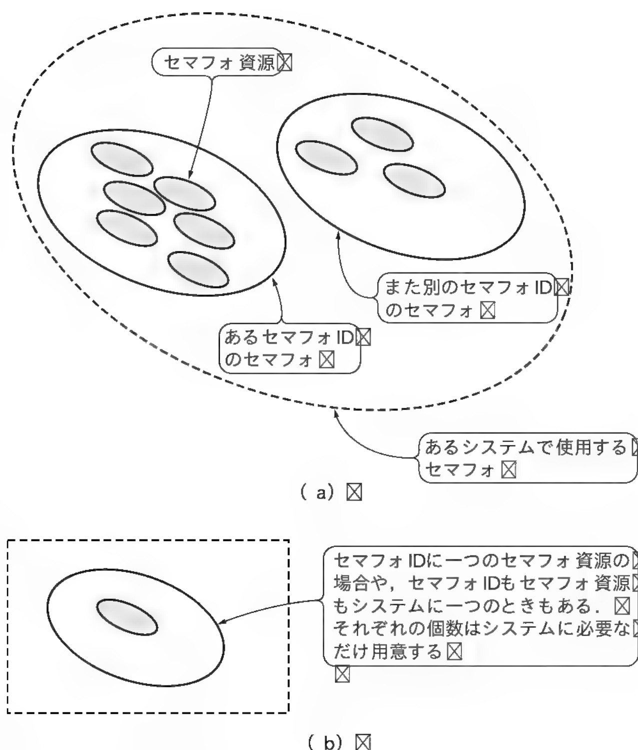


図6　セマフォの概念図2

ある/なしでわかりますし，セマフォIDが違う所に戻すことができないのと同じく，あるペンションに別のペンションの鍵を戻しても無効であることもわかると思います．

この管理対象の塊，クリティカル・セクションを排他制御するためには，入口と出口のところで，セマフォ資源の獲得と返却を行います．このとき，一度に一つのタスクからしか使えないクリティカル・セクションは，セマフォ資源の個数を，初期値，最大値ともに1として利用します．

また，生成したセマフォは複数のセマフォ資源をもつことができます．さらに，セマフォ自体も複数生成することができ，セマフォID番号で区別することができます(図6)．

少し補足しておきますが，このような図で示すと，セマフォID間でやりとりができそうに見えますが，じつはできません．つまり，あるセマフォIDは同じセマフォIDの所にしか返却できず，セマフォ資源は，セマフォID間を移ることができないのです．この制限も先ほどのペンションの例と同じです．

コラム 1 セマフォの由来

セマフォの由来は，フランス革命の最中に発明されたシャップの腕木式通信機がセマフォの由来のようです．これは腕木を柱に取り付け，回転させ，さまざまな合図を送る通信機です．この腕木式通信機の改良型が，船舶通信にも利用され，セマフォアと呼ばれました．

これが由来で，左右の手の上げ下げでアルファベットを表す国際方式の手旗信号をセマフォア信号(Semaphore Flag Signalling System)といいます．ちなみに日本独自のカタカナの手旗信号もあり，これは明治時代に考案されました．これはセマフォア信号とはいいません．

この腕木式通信機を信号機として鉄道で使用し，閉塞区間，いわゆる排他に使用したため，排他制御の処理の名前に使われるようになりました．

(出典：TDKサイエンスミュージアム)

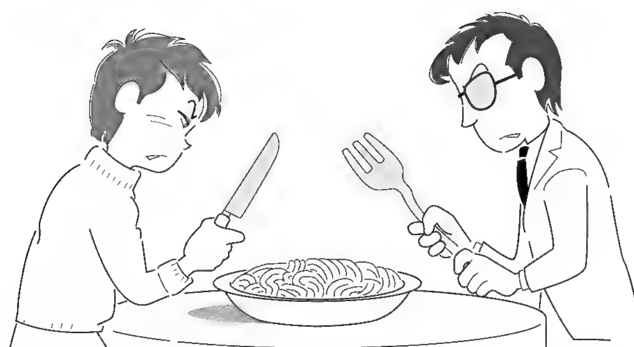


図7　ナイフとフォーク——二人で一度に二つのリソースが必要な場合

● セマフォの構成

セマフォを扱うサービス・コールには，セマフォ資源の返却を行う`sig_sem`一つに対し，獲得待ちを行う`wai_sem`，獲得を待たずにポーリングを行う`pol_sem`，タイムアウト付きで待つ`twai_sem`の三つがセットで提供され，獲得の方法を選べるようになっています．なお，JSPカーネルには含まれませんが，μITRON4.0仕様には，セマフォの状態を参照する`ref_sem`もあります．

● 複数のセマフォ，一つのセマフォ，セマフォに優先順位を持たせる

一つのクリティカル・セクションを排他制御するためには，セマフォ資源を一つ用意します．複数のクリティカル・セクションを管理する場合で，お互いに関連のない場合は，複数のセマフォを使用します．このとき，複数のセマフォはセマフォID番号を利用して区別します．

逆に，複数個あるクリティカル・セクションを一連の処理の中で使用する場合などでは，一つのセマフォ資源で複数のクリティカル・セクションを管理する場合もあります．

図7のように，二人で一度に二つのリソース(ナイフとフォーク)が必要な場合，とくに，それぞれ初めに別のリソースを取るようになった場合などは，セマフォ資源で管理しようとしても管理できません．このような場合には，デッドロックという二人のにらみ合い状態が発生します．このデッドロックを英文

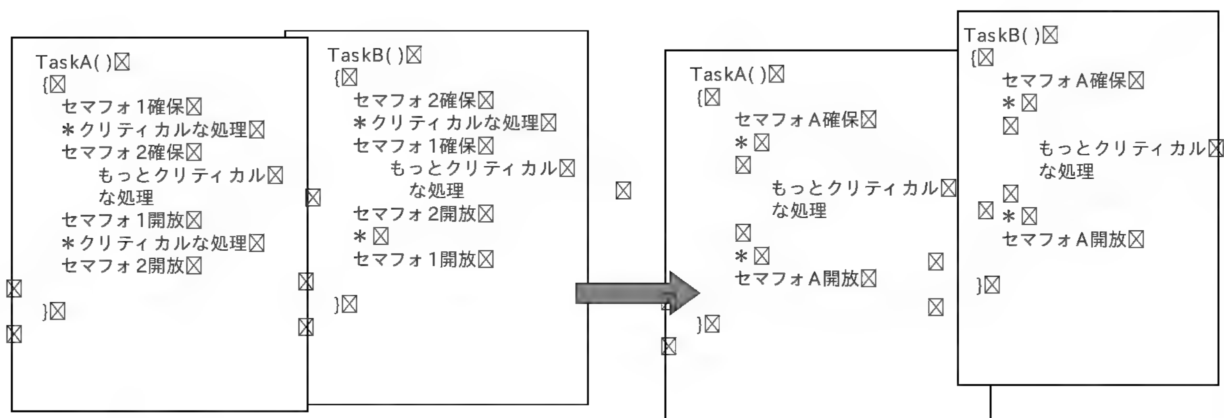


図8　セマフォの位置変更

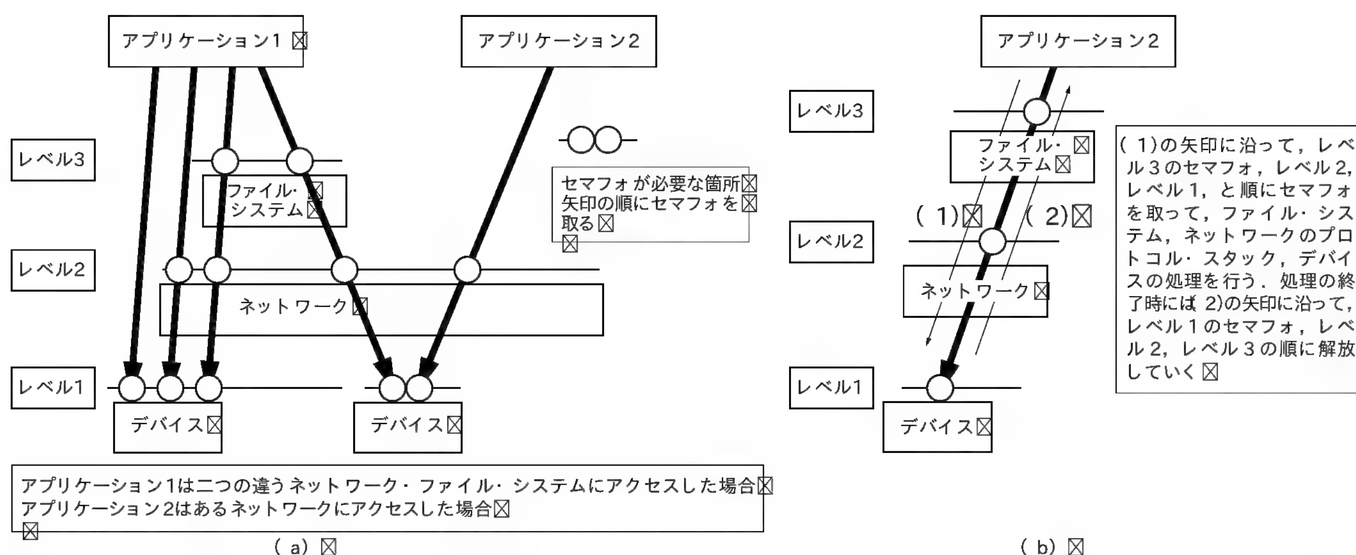


図9　ネットワーク・ファイル・システムにおけるセマフォ

の解説文書などでは死の接吻(kiss of death)などと表現することもあります．このような場合，どうやって解決すればよいでしょうか？

▶ 解法1：一つにまとめる

このような場合，ナイフとフォークを一つのリソースに見立てて管理します(図8)．セマフォ資源が一つになるので，デッドロックは発生しません．ただし，リソースを占用する時間は相対的に長くなります．

▶ 解法2：レベル分けする

ナイフとフォークの場合は良かったのですが，排他制御を一つにまとめると，問題を起こすことがあります．

たとえば，ネットワーク・ファイル・システムを構成するために，従来ネットワークのプロトコル・スタック内，ファイル・システムのドライバなどの内部で個別にセマフォを使用していた場合などです(図9)．

ここでは，たとえば，フォークにあたるセマフォがファイルシステム，ナイフにあたるセマフォがネットワークにあたります．これらは別々に扱わないと，ネットワーク・ファイル・システムを使っていると，ネットワークが使えなくなる時間が長くなり，パフォーマンスに相当な悪影響を与えます．

また，二つのネットワーク・ファイル・システムを扱うときに，セマフォを取る順序が逆になっていた場合は，デッドロックに陥ってしまいます．

このような場合，セマフォに確保する順序をもたせて，優先順位をつける解決方法があります．まず確保するセマフォを「レベル1」と「レベル2」などとレベル分けします．そして，かならず特定の順序(ここではレベル2，1の順)でセマフォを確保することで回避できます．

● セマフォの造りとそのほかの使用方法

じつは，セマフォはカウンタで実現できます(図10)．セマフォを利用したときのカウンタのふるまいは，次のようになっています．

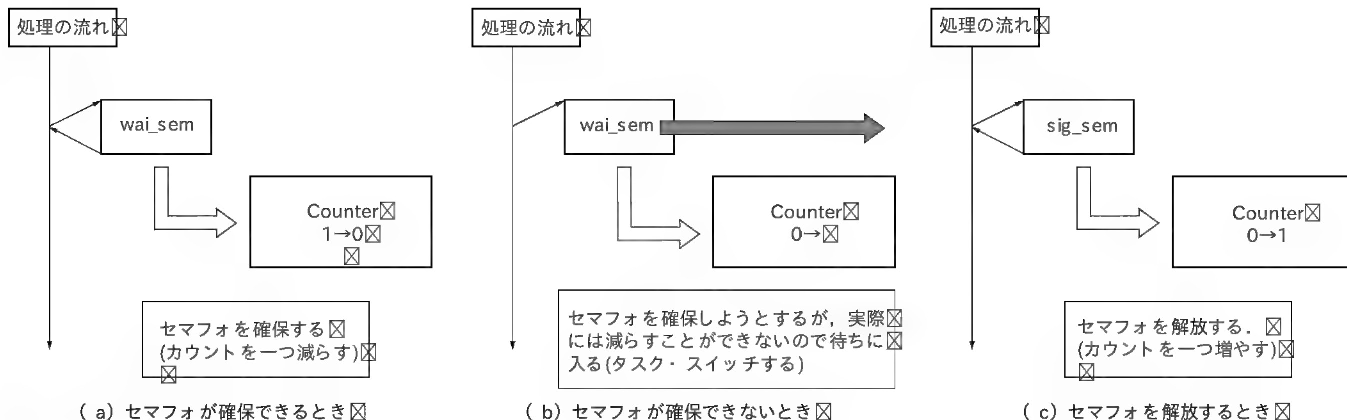


図10　セマフォ・カウンタの図1

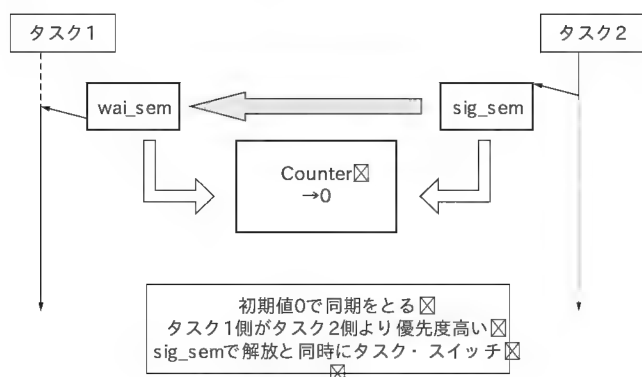


図11　セマフォ・カウンタの図2

リスト 1　Linux用ノンブロッキングI/Oサポート・モジュールより

```
CRE_SEM(SEM_LINUX_SIGIO, { TA_TPRI, 1, 1 });
```

1) wai_semでセマフォ資源の獲得に成功すると，セマフォ内のカウントは1減らされる
2) 逆にsig_semでセマフォ資源を返却するとカウンタは1増やされる

このことを利用して，セマフォは排他制御以外にも利用できます．たとえば，イベントの発生回数を伝えるカウンタです．イベントが発生するたびにsig_semシステム・コールでカウンタを加算していくと，イベントを受けるタスクはセマフォ資源を取得できるまで待ち，セマフォ資源が取得できると予定の処理を実施できることになります(図11)．

ただし，通常は一つのセマフォでカウントできる数は決まっており，TOPPERS/JSPカーネルでは，静的APIで定義したmaxsem以下までがカウント可能です．ちなみに，このセマフォに設定できる最大資源数は，コンパイラやCPUに依存しており，UINT型(unsigned int型に定義している)で表現できる値，たとえば32ビットの場合は($2^{32}-1$)＝4294967295，16ビットの場合は($2^{16}-1$)＝65535になります．

あと補足ですが，μITRON4.0仕様書には，TMAX_MAXSEMの定義が出てきますが，スタンダード・プロファイルは定義する必要がないため，TOPPERS/JSPもあわせて，この定義は行っていません．

そのほかにも，たとえば，元々のカウンタを0にしておくと，かならずwai_semで止まり，sig_semを実行するタイミングと同期を取ることもできます．

● セマフォのサービス・コール

実際にセマフォを利用するには，静的APIかC言語APIを使用して生成する必要がありますが，スタンダード・プロファイルに対応しているJSPカーネルでは，セマフォの生成は静的APIであるCRE_SEMを用いて行います．

セマフォの生成

静的API	
CRE_SEM(ID semid, { ATR sematr, UINT isemcnt, UINT maxsem });	
パラメータ	
semid	セマフォのID
sematr	セマフォの属性
isemcnt	セマフォの初期値
maxsem	セマフォの最大値

セマフォの生成は静的API，CRE_SEMを利用します．コンフィギュレーション・ファイルにリスト1のように記述します．

▶ セマフォのID

コンフィギュレーション・ファイル内に静的APIを記述し，semidにてIDの名前を指定します．

▶ セマフォの属性

セマフォの属性としてTA_TFIFOとTA_TPRIが設定可能です．TA_TFIFOはセマフォ資源の獲得待ちにあるタスクを，待ち行列にFIFO順で並べ，待ちに入った順にセマフォ資源を獲得できるようにします．TA_TPRIはタスクの優先度順に並べます．

▶ セマフォの初期値，セマフォの最大値

セマフォの初期値と最大値を指定できます．セマフォの最大値は，生成時にあるIDのセマフォに対していくつという形で，セマフォの個数を定義します．前述の最大個数まで必要な個数をセマフォで確保できます．

セマフォ資源の返却

C言語API
ER sig_sem(ID semid); ER isig_sem(ID semid);
パラメータ
ID semid 資源返却対象のセマフォID番号
返り値
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS),(E_NOSPT),(E_RSFN),E_CTX,(E_MACV), (E_OACV),(E_NOMEM),E_ID,(E_NOEXS),E_QOVR E_CTX : コンテキスト・エラー E_ID : 不正ID番号 E_QOVR : キューイング・オーバフロー

セマフォ資源を一つ返却します．返却により，もしセマフォ資源待ちのタスクがある場合は実行可能状態になります．

セマフォ資源待ちのタスクがなければ，利用可能なセマフォ資源が一つ増えます．CR_SEMで定義した最大値以上に返却しようとするとE_QOVRが返ります．セマフォIDが存在しない場合にはE_IDが返り，誤ったコンテキストで使用されるとE_CTXを返します．

ここで，誤ったコンテキストで使用した場合は，割り込みハンドラ内でiの付かないsig_semを使用した場合や，逆にタスクの処理内でiの付くisig_semを使用した場合が該当します．

セマフォ資源の獲得

C言語API
ER wai_sem(ID semid);
パラメータ
ID semid 資源獲得対象のセマフォID番号
返り値
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM), E_ID, (E_NOEXS), (E_PAR), E_RLWAI, (E_TMOUT), (E_DLT)

セマフォ資源を一つ獲得します．もしセマフォ資源がある場合は，セマフォ資源を一つ取って，実行状態のまま処理を続行します．セマフォ資源がなければ，待ち状態に入ります．

セマフォの属性としてTA_TFIFOを指定した場合は，待ち行列の最後に並べ(FIFO順になります)，TA_TPRIが指定されていた場合は，優先度を比較しながら待ち行列につなぐ位置を決めます(タスクの優先度順に並びます)．

セマフォIDが存在しない場合は，E_IDが戻り，sig_semと同じく，誤ったコンテキストで使用されるとE_CTXを返します．また，サービス・コールrel_waiを用いて，待ち状態から強制解除された場合は，E_RLWAIを返します．

セマフォ資源の獲得(ポーリング)

C言語API
ER pol_sem(ID semid);
パラメータ
ID semid 資源獲得対象のセマフォID番号
返り値
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM), E_ID, E_NOEXS, E_PAR, E_RLWAI, E_TMOUT, E_DLT

セマフォ資源を一つ獲得します．もしセマフォ資源がある場合は，セマフォ資源を一つ獲得して，実行状態のまま処理を続行します．セマフォ資源がなくても待ち状態に入りません．セマフォ資源獲得のため，ポーリングを行う場合に利用します．

たとえば，セマフォ資源が確保できなかった場合は別処理を行うなど，セマフォを獲得するために時々確認を行う場合などに使用します．セマフォ資源が獲得できなかった場合には，E_TMOUTを返します．

セマフォ資源の獲得(タイムアウトあり)

C言語API
ER twai_sem(ID semid, TMO tmout);
パラメータ
ID semid 資源獲得対象のセマフォID番号 TMO tmout タイムアウト指定
返り値
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM), E_ID, E_NOEXS, E_PAR, E_RLWAI, E_TMOUT, E_DLT E_PAR：パラメータ・エラー

セマフォ資源を一つ獲得します．もしセマフォ資源がある場合は，セマフォ資源を一つ取って，実行状態のまま処理を続行します．セマフォ資源がなければ，一定時間待ち状態に入ります．一定時間待っても，セマフォ資源が獲得できなかった場合にはE_TMOUTを返します．

待ち状態のときにrel_waiが発行された場合はE_RLWAIを返し，TMO_FEVRより長い時間を指定したときはE_PARを返します．

● **実際に使ってみる(sig_sem，wai_sem)**

セマフォ資源を確保する所をリスト2に示します．タスクの動作は図12を参考にしてください．

さらにサンプル・ソースを変更してみましょう．リスト2を元に修正し，リスト3のようにしてみました．二つのセマフォを獲得します．三つ以上のセマフォを獲得する場合も同じです(リスト4)．

図12は，先に説明していたように排他制御の範囲を広げる場合の例です．排他制御の範囲を広げているので，排他制御の対象となっているリソースを獲得できない最悪値は長くなります．

● 状態遷移図

あるタスクにて，セマフォ待ちに入ると，図1(p.99)のように待ち状態に状態遷移します．

この待ち状態に入ること自体は，イベント・フラグや，データ・キューなどを使用している場合も同じです．ただし，イベント・フラグや，データ・キューなどで待ち状態にあるタスクを強制待ち状態に入れることはできても，セマフォを使用している場合は待ち状態に入れることができません．セマフォをもったまま強制待ち状態へ遷移させたりしないような注意が必要です．

● デッドロックさせてみる

では，理解を進めるために，あえてデッドロックさせてみましょう．

リスト5，図13は，先に説明したデッドロックする例です．デッドロックさせてみると，セマフォ資源を取り合いになったタスクの二つがセマフォ待ちの状態で停止します．そのほかのタスクは動作しており，画面にはメイン・タスクのログが定期的に出力されます．

実際のアプリケーションで，デッドロックさせてしまった場合にも，このように特定のタスクのみ停止し，ほかのタスクは動作している状態になります．このため，見かけ上は一部の機

リスト2　シンプルな例

```
/*
 *  イベント通知のタスク
 */
void task1(VP_INT exinf)
{
    while(1) {
        set_flg(FLAG_ID, 0x1);      /* イベント通知 */
        proc_something();
    }
}
/*
 *  起床されるタスク
 */
void task2(VP_INT exinf)
{
    /* イベント発生時のフラグの値をコピーする領域 */
    FLGPTN flgptn;

    while(1) {
        /* イベント待ち(WAITING状態へ)  */
        wai_flg(FLAG_ID, 0x1, 0x0, &flgptn);
        proc_something();
    }
}
```

リスト3　二つのセマフォ

```
void task1(VP_INT exinf)
{
    while(1) {
        /* 資源待ち(資源がなければWAITING状態へ)  */
        wai_sem(SAMAPHORE_ID1);
        /* 資源待ち(資源がなければWAITING状態へ)  */
        wai_sem(SAMAPHORE_ID2);
        proc_something();
        sig_sem(SAMAPHORE_ID2); /* 資源返却 */
        sig_sem(SAMAPHORE_ID1); /* 資源返却 */
    }
}
```

リスト4　三つのセマフォ

```
void task1(VP_INT exinf)
{
    while(1) {
        /*一つめのリソースを確保*/
        /* 資源待ち(資源がなければWAITING状態へ)  */
        wai_sem(SAMAPHORE_ID1);
        /*二つめのリソースを確保*/
        /* 資源待ち(資源がなければWAITING状態へ)  */
        wai_sem(SAMAPHORE_ID2);
        /*三つめのリソースを確保*/
        /* 資源待ち(資源がなければWAITING状態へ)  */
        wai_sem(SAMAPHORE_ID3);

        proc_something2();


        sig_sem(SAMAPHORE_ID3); /* 資源返却 */
        sig_sem(SAMAPHORE_ID2); /* 資源返却 */
        sig_sem(SAMAPHORE_ID1); /* 資源返却 */
    }
}
```

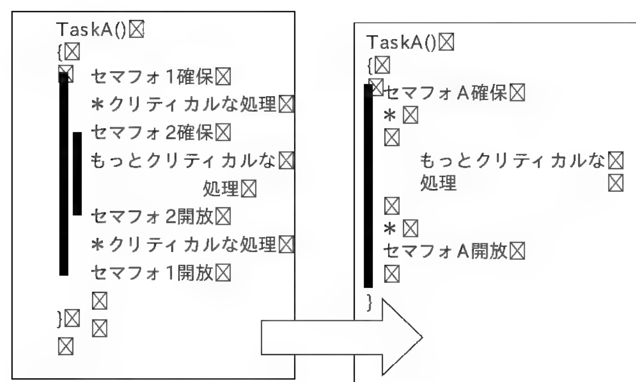


図12　排他制御の範囲を広げる

リスト5　デッドロックをさせてみるプログラム

```
void task1(VP_INT exinf)
{
    while(1) {
        /* 資源待ち(資源がなければWAITING状態へ)  */
        wai_sem(SAMAPHORE_ID1);
        /* 資源待ち(資源がなければWAITING状態へ)  */
        wai_sem(SAMAPHORE_ID2);
        proc_something();
        sig_sem(SAMAPHORE_ID2); /* 資源返却 */
        sig_sem(SAMAPHORE_ID1); /* 資源返却 */
    }
}

void task2(VP_INT exinf)
{
    while(1) {
        /* 資源待ち(資源がなければWAITING状態へ)  */
        wai_sem(SAMAPHORE_ID2);
        /* 資源待ち(資源がなければWAITING状態へ)  */
        wai_sem(SAMAPHORE_ID1);
        proc_something();
        sig_sem(SAMAPHORE_ID1); /* 資源返却 */
        sig_sem(SAMAPHORE_ID2); /* 資源返却 */
    }
}
```

能のみ異常な状態に陥ることになります．たとえば，ロックしてしまった片方のタスクが，処理のループのなかで割り込み処理などから起床要求を受け取るなど行っていたとします．これは，割り込み処理から要求された処理が行われないなどの症状になります．このため，見方やタイミングによっては，正常に動作しているように見えてしまいます．

このような場合の対処法に関しては，正常に動作している部分のほうが多かったりすると，順に切り分ける方法では，解決までに時間が掛かる場合もあります．推測と検証を順に行う方法では，思いつきによる問題解決に陥りやすいので，勧められません．しかし，セマフォを扱う場合には，例外的に試行しても良いのではないかと思います．

リスト5のように見ただけで問題になる箇所が特定できるのであれば，すぐに気付くでしょう．しかし実際には，いにしえのソフトウェアを綿々と保守している環境など，相当深い階層構造などにはばまれて見つけ出すことが困難な場合があります．

このような問題の回避には，どうすれば良いでしょうか．筆者の思うところは次のような点です．まず，程度の問題ですが，ソースをわかりやすい状態に置いておく必要があるかと思います．ここではソースをわかりやすい状態に維持するために，以下のことを実践されることを勧めます．

▶ソースをわかりやすい状態に維持するためにシンプルな作りに維持する

同じ階層の関数でセマフォ資源を獲得，返却する．セマフォ資源を持ったままの，大域脱出，タスク終了はしない．

▶あるパターンに従う

たとえば，関数の入口と出口でセマフォを獲得，返却する対になるように，もう一方を推測できる位置におく．

▶ドキュメント，コメントを入れておく

排他制御の範囲はかならずドキュメントに残す．類似点の洗い出しは資料に残す．機能拡張時は，排他制御の範囲，類似点など，修正範囲の確認を行う．

● 設計で抑えよう

デッドロックに陥っているかどうかを判断する場合，スケジューラが管理するTCB(タスク・コントロール・ブロック)を確認すればすぐにデッドロックに陥っているとわかります．デッドロックを疑うような不具合が発生した場合，タスクがどのような状態にあるかどうかを把握するのは難しいため，関連するタスク全部をソース・コード上から追いかけることにもなりかねません．

デッドロックを発生させないようには，設計のレビューなどで発生を抑えておくべきです．対象となる製品の規模にもよりますが，以下四つの項目を押さえておくことを勧めます．

1) 使用しているセマフォのID一覧
2) セマフォ資源を取得する順序
3) 競合する場合の条件(いつ，どこで)
4) 機能追加した場合は従来との整合性

イベント・フラグ

● イベント・フラグの概要

イベント・フラグとは，ビット・パターンを使って，イベントの有無を通知するサービス・コールです．このビット・パターンはJSPカーネル内の変数(フラグと呼ぶ)に保持されてい

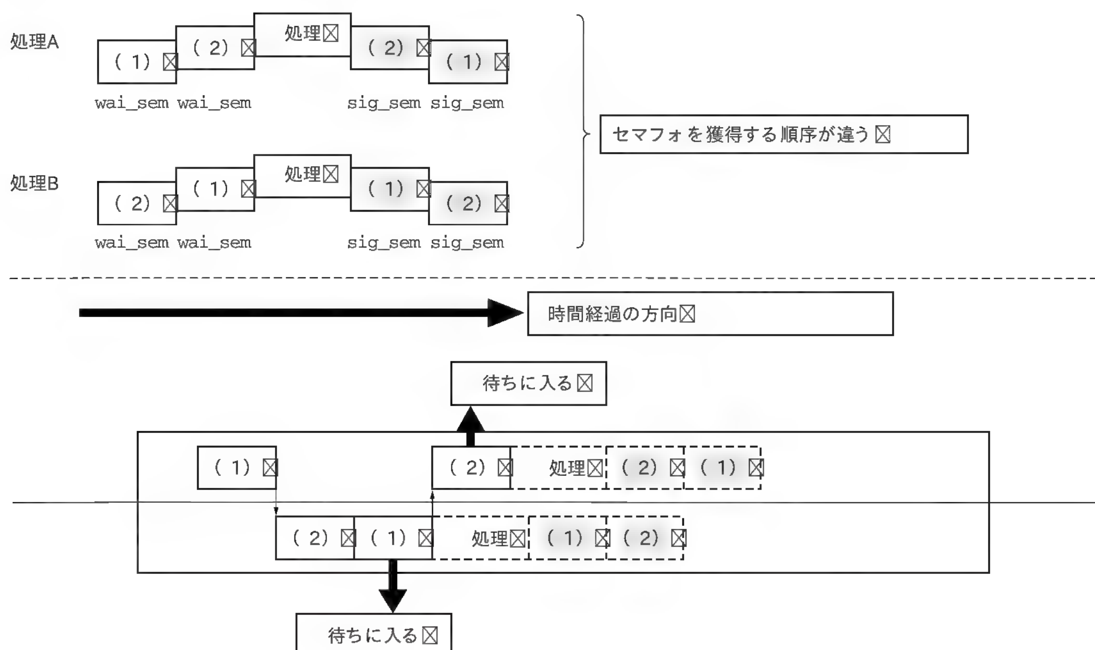


図13　デッドロックの例

ます．どのビットにどういう意味をもたせるかは設計者が決めます．ビット・パターンを指定することで，待ち条件を指定でき，同期を取ることができます．

なお，イベント・フラグは，立ったときにイベントを通知できるもので，レベル(0か1か)や，クリアされたことの通知はできません．

カーネル内では，ビット・パターンを書き込んだ直後に，条件が成立したかどうかを判断し，成立した場合は，待ち状態にあったタスクを実行可能状態に移します．このとき，優先度がいちばん高ければ，このタスクは実行状態に移ります．

イベント・フラグのサービス・コールにはビット・パターンをセットするset_flg，iset_flgがあり，イベントの発生を待つ方法によってwai_flg，pol_flg，twai_flgが利用できます．イベント・フラグをクリアするサービス・コールとしてclr_flgも用意されています．ただし，イベント・フラグのクリアには待ち条件成立時にクリアする方法と，サービス・コールclr_flgを用いる方法があります．

これらサービス・コールの利用方法は図14のようになります．JSPカーネル内のフラグの変化を待つタスク，フラグに特定のビット・パターンを設定するタスクからなります．

待ち条件としては，TWF_ANDW(AND条件待ち)，TWF_ORW(OR条件待ち)が指定でき，それぞれ以下の意味を持ちます

- TWF_ANDW(AND条件待ち)：イベント・フラグの待ちビット・パターンがすべて成立するまで
- TWF_ORW(OR条件待ち)：イベント・フラグの待ちビット・パターンのどれかが成立した場合

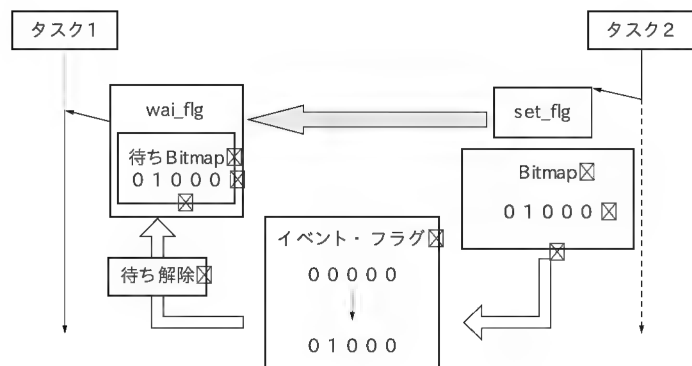


図14　サービス・コールの利用方法

リスト6　イベント・フラグの生成

```
#define FLAG_INIT (FLGPTN)0
CRE_FLG(FLAG_ID, {TA_CLR, FLAG_INIT});
```

イベント・フラグの生成

イベント・フラグを使用するためにはイベント・フラグの生成する必要があります．イベント・フラグの生成は静的API，CRE_FLGを使用します．

静的API	
CRE_FLG(ID flgid,{ATR flgatr,FLGPTN iflgptn});	
パラメータ	
flgid	イベント・フラグのID
flgatr	イベント・フラグの属性
iflgptn	イベント・フラグの初期値

● イベント・フラグのID

コンフィギュレーション・ファイル内に静的APIを記述し，flgidに，ID番号を割り振ってほしい名前を指定します(リスト6)．コンフィグレータは，この名前からイベント・フラグのID番号を割り振ったヘッダ・ファイルを生成します．このヘッダ・ファイルに，記号定数としてflgidに記述したID名が定義されます．

● イベント・フラグの属性

μITRON4.0仕様では，イベント・フラグの属性として，2種類を指定できます．複数のタスクが待ちに入ることを許す属性(TA_WMUL)と，一つのタスクしか待ちに入ることができない属性(TA_WSGL)です．ただし，JSPカーネルでは，仕様上，複数のタスクが待ちに入ることはできず，TA_WSGLの指定のみ許されています．

属性としては，イベント・フラグの待ち行列もFIFO順(TA_FIFO)か，タスクの優先度順(TA_TPRI)かを指定できます．これも，JSPカーネルでは複数のタスクが待ちに入ることができないため，結果的に同じ動作になります．

また，表2にイベント・フラグの属性として指定可能な設定値を示します．JSPカーネルでは，表2より指定可能で有効な設定はTA_WSGLのみとわかってもらえると思います．

● イベント・フラグの初期値

初期値として使用したいビット・パターンを指定します．方法は，静的APIを記述するときにiflgptnにビット・パターンを指定します．

このビット・パターンは，最初から立っているビット・パターンをカーネル内部にあるJSPカーネル内の変数(フラグ)に

表2
イベント・フラグの属性

TA_FIFO	待ち行列がFIFO順	0x00	指定可能(ただし，実際のふるまいに違いはない)
TA_TPRI	待ち行列がタスクの優先度順	0x01	指定可能(ただし，実際のふるまいに違いはない)
TA_WSGL	一つのタスクだけ待ちに入る	0x00	指定可能，かつ有効な設定
TA_WMUL	複数タスクが待ちに入る	0x02	スタンダード・プロファイルではサポートする必要なし
TA_CLR	待ち解除時にフラグ・クリア	0x04	

注：ビット・パターンなので0，1，2，4となっている

初期値として与えられます．初期値を与えることにより，イベント・フラグによるイベントの通知を，初期状態からイベントが発生したように見せかけることができます．

たとえば，あるビットが立つのを待つタスクがあるとすると，あるビットを立てて初期化した場合，システムの初期化直後もタスクは待ち状態に入りません．そして，あたかもイベントが発生したように扱われます．

イベント・フラグのセット

C言語API
ER set_flg(ID flgid, FLGPTN setptn); ER iset_flg(ID flgid, FLGPTN setptn);
パラメータ
ID flgid　イベント・フラグID FLGPTN setptn イベント・フラグにセットするビット・パターン
返り値
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM), E_ID, (E_NOEXS), (E_PAR)

指定したビット・パターンをイベント・フラグに設定します．設定は現在のイベント・フラグの内容にORする形で格納されるので，すでにセットされているビット・パターンをセットした場合は，タスクに関する状態変化は発生しません．これは，すでにビット・パターンがセットされていることから，待ち状態にあるタスクの解除条件と関連しないビット・パターンといえます．

イベント・フラグのクリア

C言語API
ER clr_flg(ID flgid, FLGPTN clrptn);
パラメータ
ID flgid　イベント・フラグID FLGPTN clrptnイベント・フラグにからクリアするビット・パターン
返り値
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM), E_ID, (E_NOEXS), (E_PAR)

指定したイベント・フラグのビット・パターンをクリアします．クリアする方法は，指定したクリア・パターンとイベント・フラグの内容でANDを取り，再度イベント・フラグに結果を保管します．このため，クリアしたいビットを0に，クリアしたくないビットには1を指定する必要があります

イベント・フラグ待ち

C言語API
ER wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);
パラメータ
ID flgid　イベント・フラグID FLGPTN waiptn　待ちビット・パターン MODE wfmode　待ちモード
返り値
FLGPTN *p_flgptn　待ち解除時のビット・パターン ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM), E_ID, (E_NOEX), E_PAR, E_ILUSE, E_RLWAI, (E_TMWAI), E_TMOUT, (E_DLT) E_ILUSE：サービス・コール不正使用 E_RLWAI：待ち状態の強制解除

指定した待ち条件とビット・パターンで待ち状態に入ります．JSPカーネルでは一つのタスクしか待ちに入れないので，すでに同じイベント・フラグIDで待ち状態にあるタスクがいる場合，E_ILUSEを返します．

指定できない，待ちビット・パターン(waiptn)か，待ちモード(wfmode)の値を指定した場合，E_PARを返します．待ちビット・パターンでエラーとなるのは，設定できない待ちパターン，どのビットも待たない0を指定したときのみです．このビット・パターンでは待ち解除を行うことができません．wai_flg，pol_flg，twai_flgとも，使用する場合には，どれかのビットが立っていて，待ち状態から解除できる，0以外のパターンを指定してください．

待ちモードでエラーとなるのは，AND待ちの指定(TWF_ANDW)かOR待ちの指定(TWF_ORW)以外の誤った指定を行ったときのみで，通常はありえません．なお，AND待ちかOR待ちの指定かは，どちらかのみが許されています．誤って

両方指定してしまった場合には，OR 待ちの指定(TWF_ORW)が優先されます．

イベント・フラグ待ち（ポーリング）

C言語API	
ER pol_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn);	
パラメータ	
ID flgid	イベント・フラグ ID
FLGPTN waiptn	待ちビット・パターン
MODE wfmode	待ちモード
FLGPTN *p_flgptn	待ち解除時のビット・パターン
返り値	
ER E_OK(正常終了)またはエラー・コード	
エラー・コード	
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM), E_ID, (E_NOEXS), E_PAR, E_ILUSE, E_RLWAI, (E_TMWAI), E_TMOUT, (E_DLT) E_TMOUT：ポーリング失敗またはタイム・アウト	

pol_flg は wai_flg の処理をポーリングで行います．待ちに入らず，イベントがすでに発生しているかどうかを確認したい場合に使用します．イベントが発生していない場合は，E_TMOUT が返ります．

イベント・フラグ待ち（タイムアウトあり）

C言語API	
ER twai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout);	
パラメータ	
ID flgid	イベント・フラグ ID
FLGPTN waiptn	待ちビット・パターン
MODE wfmode	待ちモード
FLGPTN *p_flgptn	待ち解除時のビット・パターン
返り値	
ER E_OK(正常終了)またはエラー・コード	
エラー・コード	
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM), E_ID, (E_NOEXS), E_PAR, E_ILUSE, E_RLWAI, (E_TMWAI), E_TMOUT, (E_DLT) E_RLWAI：待ち状態の強制解除 E_TMOUT：ポーリング失敗またはタイム・アウト	

twai_flg は wai_flg の処理をタイム・アウト付きで行います．待ちには入りますが，指定時間内にイベントが発生しなかった場合に制御を戻したいときに使用します．指定時間内にイベントが発生しなかった場合は，E_TMOUT が返ります．

なお，タイム・アウトの指定引き数(tmout)に TMO_POL を指定すると，pol_flg と同じふるまいになり，同じく TMO_FEVER を指定すると wai_flg と同じふるまいになります．

● **実際に使ってみる(set_flg，wai_flg，clr_flg)**

優先順位とイベント・フラグの操作をタスクのふるまいから見てみましょう．タスクの動作を示すと図 15 のようになります．当たり前ですが，優先度の高いタスクの実行中の間は，イベント・フラグに変化があってもタスク・スイッチは発生しません．

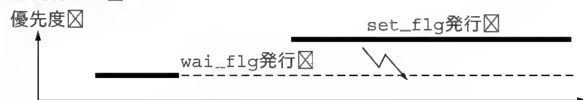
優先度が低いタスクでイベント・フラグの待ちに入った場合に，優先度が高いタスクがイベント・フラグをセットし，イベントを発生させてもタスク・スイッチはされず，優先度の高いタスクの処理が続行されます．

リスト 7の処理で，優先度が高いタスクがイベント・フラグをセットしてもタスク・スイッチはされないのですが，dly_tsk などで待ちに入った場合は，優先度の低いタスクの処理へとスイッチし，優先度の低いタスクの処理が行われます．当たり前ですが，優先順位の低いタスクを処理するには，優先度の高いタスクが待ちに入る，または，act_tsk，ext_tsk を使うなどの手段で終了させる必要があります．割り込み処理からの iset_flg を使用した場合も同じふるまいになります(リスト 8)．

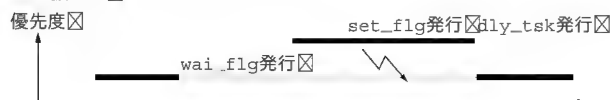
次に同じ優先度のタスク間でのふるまいを見てみましょう(リスト 9)．

逆のパターンとして優先度が低いタスクから高いタスクにイベントを通知した例です(リスト 10)．

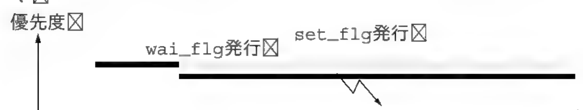
優先度の低いタスクでwai_flgを発行し，優先度の高いタスクからset_flgを発行する



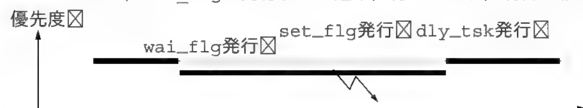
優先度の低いタスクでwai_flgを発行し，優先度の高いタスクからset_flgを発行する．その後dly_tskを発行すると，優先度の低いタスクは待ち状態から抜ける



同一優先度のタスクが二つあり，wai_tskを発行した．同一優先度のタスクにスイッチし，set_flgを発行するが，待ちに入らず，そのまま実行が続く



同一優先度のタスクが二つあり，wai_tskを発行した．同一優先度のタスクにスイッチし，set_flgを発行する．途中，待ちに入り，制御が移る



優先度高のタスクから，wai_tskを発行した．優先度低のタスクが実行され，set_flgを発行する．発行直後，優先度高のタスクにスイッチする

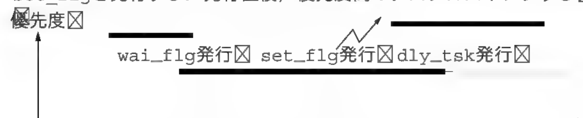


図 15　タスクの動作――優先度とふるまい

リスト7　優先度の低いタスクの処理1

```
sample1.h
---
#define MAIN_PRIORITY   5       /* メイン・タスクの優先度 */
                    /* HIGH_PRIORITY より高くすること */

/* 並列に実行されるタスクの優先度 */
#define HIGH_PRIORITY   9
#define MID_PRIORITY    10
#define LOW_PRIORITY    11

------------------------------------------------------------
sample1.cfg
--
CRE_TSK(TASK_ID1, { TA_HLNG, (VP_INT) 1, input_task,
                      MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK_ID2, { TA_HLNG, (VP_INT) 1, output_task,
                      LOW_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK_ID3, { TA_HLNG, (VP_INT) 1, control_task,
                      MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(MAIN_TASK, { TA_HLNG|TA_ACT, 0, main_task,
                      MAIN_PRIORITY,
            STACK_SIZE, NULL });

------------------------------------------------------------
sample1.c
--
/*
 *  優先度が高いタスク
 *  入力タスク
 */
void input_task(VP_INT exinf)
{
    /* イベント発生時のフラグの値をコピーする領域 */
    FLGPTN flgptn;

    while(1) {
        syslog_0(LOG_NOTICE, "入力タスク動作中です");
        syslog_0(LOG_NOTICE,
       "入力タスク イベント待ち(WAITING状態へ)に入ります");

        /* イベント待ち(WAITING状態へ) */
        wai_flg(FLAG_ID1, (BIT_ACT1|BIT_ACT3), TWF_ORW,
                  &flgptn);
        syslog_1(LOG_NOTICE,
             "入力タスク イベント flg=%x を受け付け",flgptn);

        syslog_0(LOG_NOTICE,
             "入力タスク 出力タスクへイベントを通知します");
        set_flg(FLAG_ID2, BIT_ACT2);

        while(1); /* ★ */

    }
}
/*
 *  優先度が低いタスク
 *  出力タスク
 */
void output_task(VP_INT exinf)
{
    /* イベント発生時のフラグの値をコピーする領域 */
    FLGPTN flgptn;

    while(1) {
        syslog_0(LOG_NOTICE, "出力タスク動作中です");
        syslog_0(LOG_NOTICE,
       "出力タスク イベント待ち(WAITING状態へ)に入ります");

        /* イベント待ち(WAITING状態へ) */
        wai_flg(FLAG_ID2, BIT_ACT2, TWF_ORW, &flgptn);
        syslog_1(LOG_NOTICE,
             "出力タスク イベント flg=%x を受け付け",flgptn);

        syslog_0(LOG_NOTICE,
                    "出力タスク 再度イベント待ちに入ります");
        /* イベント待ち(WAITING状態へ) */
        /* あてのないイベント待ち */
            /* 以下の入れ替え用とどちらかを使う */
        wai_flg(FLAG_ID3, BIT_ACT2, TWF_ORW, &flgptn);

        /* 入れ替え用 */
        //syslog_0(LOG_NOTICE,
                          "出力タスクから 入力タスクを起動");
        //set_flg(FLAG_ID1, BIT_ACT3);

    }
}
/*
 *  優先度が高いタスク
 *  管理タスク
 */
void control_task(VP_INT exinf)
{
        set_flg(FLAG_ID1, BIT_ACT3);
}
```

リスト9　同一優先度のタスクの処理

```
sample1.cfg
--
CRE_TSK(TASK_ID1, { TA_HLNG, (VP_INT) 1, input_task,
                      MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK_ID2, { TA_HLNG, (VP_INT) 1, output_task,
                      MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(TASK_ID3, { TA_HLNG, (VP_INT) 1, control_task,
                      MID_PRIORITY, STACK_SIZE, NULL });
CRE_TSK(MAIN_TASK, { TA_HLNG|TA_ACT, 0, main_task,
                       MAIN_PRIORITY,
            STACK_SIZE, NULL });
```

データ・キュー

● データ・キューの概要

データ・キューは，1ワード分のデータをタスク間で受け渡しする機能を提供します．リング・バッファを利用してこの機能を実現しているため，データの送信側(イベントの発生)，受信側(イベントの処理)の瞬間的な速度差を吸収することができ

リスト8　優先度の低いタスクの処理2

```
sample1.c
--
/*
 *  優先度が高いタスク
 *  入力タスク
 */
void input_task(VP_INT exinf)
{
    /* イベント発生時のフラグの値をコピーする領域 */
    FLGPTN flgptn;

    while(1) {
        syslog_0(LOG_NOTICE, "入力タスク 動作中です");
        syslog_0(LOG_NOTICE,
       "入力タスク イベント待ち(WAITING状態へ)に入ります");

        /* イベント待ち(WAITING状態へ) */
        wai_flg(FLAG_ID1, (BIT_ACT1|BIT_ACT3), TWF_ORW,
                  &flgptn);
        syslog_1(LOG_NOTICE,
             "入力タスク イベント flg=%x を受け付け",flgptn);

        syslog_0(LOG_NOTICE,
             "入力タスク 出力タスクへイベントを通知します");
        set_flg(FLAG_ID2, BIT_ACT2);

        dly_tsk(100); /* ★ */
    }
}
```

リスト 10　優先度の高いタスクの処理

```
/*
 *  優先度が高いタスク
 *  入力タスク
 */
void input_task(VP_INT exinf)
{
    /* イベント発生時のフラグの値をコピーする領域 */
    FLGPTN flgptn;

    syslog_0(LOG_NOTICE, "入力タスク 動作中です");
    while(1) {
        syslog_0(LOG_NOTICE,
           "入力タスク イベント待ち( WAITING状態へ) に入ります");
        wai_flg(FLAG_ID1, (BIT_ACT1|BIT_ACT3), TWF_ORW,
                  &flgptn);

        syslog_1(LOG_NOTICE,
             "入力タスク イベント flg=%x を受け付け",flgptn);

    }
}
/*
 *  優先度が高いタスク
 *  出力タスク
 */
void output_task(VP_INT exinf)
{
    /* イベント発生時のフラグの値をコピーする領域 */
    FLGPTN flgptn;

    syslog_0(LOG_NOTICE, "出力タスク 動作中です");
    while(1) {
        syslog_0(LOG_NOTICE,
                  "出力タスクから 入力タスクを起動");
        set_flg(FLAG_ID1, BIT_ACT3);
    }
}
```

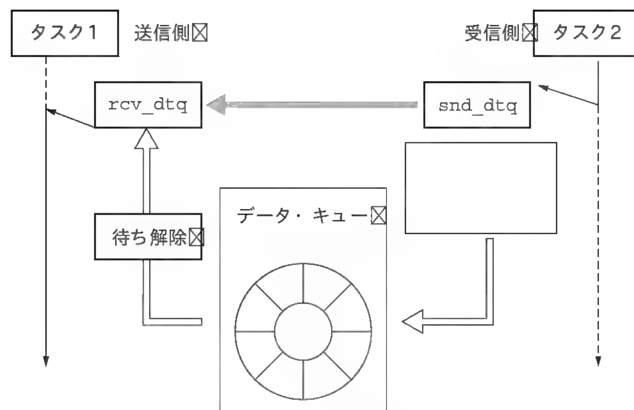


図 16　データ・キューの構造 1 ── タスク間のバッファとして使う

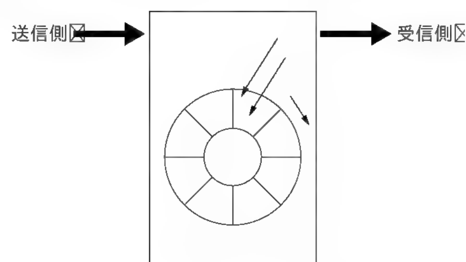


図 17　データ・キューの構造 2 ── FIFOとして使う

ます．図 16では左側に送信側，右側を受信側としていますが，このようにタスク間のバッファとして使います．

ただし，このリング・バッファの容量以上の速度差を吸収することはできません．通常は設計時に容量を算出します．これは，イベントの発生頻度，とくに最大瞬間風速のような単位時間の発生頻度と，受信側タスクの処理時間を見積もっています．このためには，送信側タスクが待ちに入っても仕方がない条件を設定してから算出します．

なお，通常は，送信側タスクが待ちにはいるとイベントの発生を通知できないので，待ちに入れないように設計するか，強制送信を使うなどとする場合もあります．

データ・キューにデータを入れることができなかった場合は，待ち行列につながれます．また，受信する場合にデータがなかった場合も待ち行列につながれます．送信時の待ち行列は，FIFO順(TA_TFIFO) かまたは，タスクの優先度順(TA_TPRI) の順を選べますが，受信時はFIFO順となります．

参考として，データ自体もFIFO順に扱われます(図 17)．

● データ・キューによる同期

データ・キューのバッファ・サイズを 0にすると，データ・キューによる同期通信を行うことができます．通常はバッファ数は 1以上のことが多いのですが，バッファ数を 0にすることで，相手タスクが受信に入っていなければ送信待ち，相手タスクからの送信がなければ受信待ちとなり，タスク間の同期を取ることができます．

それぞれ，相手タスクが受信にはいる，相手タスクが送信することで待ちから解除され，自タスク，相手タスクとも実行できる状態になります．

データ・キューの生成

静的API	
CRE_DTQ(ID dtqid, { ATR dtqatr, UINT dtqcnt, VP dtq });	
パラメータ	
ID dtqid	データ・キューID
ATR dtqatr	データ・キューの属性
UINT dtqcnt	データ・キュー領域の個数
VP dtq	データ・キュー領域の先頭番地

スタンダード・プロファイルでは，データ・キュー領域の先頭番地(dtq)はNULL 以外サポートする必要がありません．このため，JSP カーネルでもサポートしていません．

● データ・キューの ID

セマフォ，イベント・フラグと同じく，コンフィギュレーション・ファイル内に静的API を記述し，dtqidに，ID番号を割りふってほしい名前を指定します．

● データ・キューの属性

データ・キューの送信時の待ち行列の属性として，FIFO順(TA_FIFO)か，タスクの優先度順(TA_TPRI)が指定できます．データ・キュー内のデータ，および，受信待ちのタスク待ち行

列はFIFO順となっています．

● データ・キューのサンプル

データ・キューのバッファサイズを5とすると，リスト11のようになります．なお，DTQ_IDはkernel_id.hで定義されます．

リスト11　データ・キューの生成

```
#define COUNT_MAX    5
CRE_DTQ(DTQ_ID, {TA_TFIFO, COUNT_MAX, NULL});
```

● 補足：データ・キューのデータ保管順序

データ・キュー内のデータはFIFO順になっていると述べたばかりですが，特定の条件下で，FIFO順にならず，順序が変わってしまうケースがあります．これは，サービス・コールsnd_dtqとfsnd_dtqを混在させた場合に発生します．

たとえば，ABCDEFG……と順にデータが発生しているとします．図18のような状況になった場合，ABCDEFとかの順でもらうはずのデータが，ここではABEDの順となり，EがDより先に届くなど，順番が入れ替わってしまっています．

図18　レアケース発生状況

```
A B C [FULL]
    snd_dtqでD送信
      待ち行列に追加(D送信されず)
    fsnd_dtqでE送信
A B E [FULL]
    rcv_dtq
      バッファからAを受信
B E  [ひとつ空きあり]
      待ち状態のsnd_dtqの送信実行
B E D [FULL]
```

データ・キューへの送信

C言語API
ER snd_dtq(ID dtqid, VP_INT data);
パラメータ
ID dtqid　　データ・キューのID VP_INT data　データ・キューへ送信するデータ
返り値
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), (E_CTX), (E_MACV), (E_OACV), (E_NOMEM), E_ID, (E_NOEXS), (E_PAR), E_RLWAI, (E_TMOUT), (E_DLT) E_RLWAI：待ち状態の強制解除

指定したデータ・キューにデータを送信します．データ・キューへの送信に関しては複数ありますが，送信できなかった場合のふるまいに違いがあります．

snd_dtqでは送信できなかった場合は，データ・キューへの送信待ち行列に自タスクをつなげます．送信できた場合で，待ちタスクが居た場合は，待ちタスクを待ち状態から解除し，データ・キューのデータを渡します．送信できたが，待ちタスクが居ない場合は，データ・キューのバッファ内にデータを保管します．

なお，命名規則から見ると，割り込み処理で使用するisnd_dtqがありそうに思えますが，これは存在しません．割り込み処理内での使用を考えると，待ち状態に入らない(ipsnd_dtq)，強制的に送信する(ifsnd_dtq)のどちらかを行う必要があります．このため，用途に合わせてisnd_dtqを二つに分けたと覚えてもらえれば良いと思います．

ipsnd_dtq，ifsnd_dtqのサービス・コールの解説は後述しています．

データ・キューへの送信(ポーリング)

C言語API
ER psnd_dtq(ID dtqid, VP_INT data); ER ipsnd_dtq(ID dtqid, VP_INT data);
パラメータ
ID dtqid　　データ・キューのID VP_INT data　データ・キューへ送信するデータ
返り値
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), (E_CTX), (E_MACV), (E_OACV), (E_NOMEM), E_ID, E_NOEXS, E_PAR, E_RLWAI, E_TMOUT, E_DLT E_TMOUT：ポーリング失敗

指定したデータ・キューにデータを送信します．psnd_dtqは，送信できなかった場合でも送信待ち状態に入りません．送信できなかった場合はポーリング失敗(E_TMOUT)を返します．

送信できた場合は，snd_dtqと同じふるまいになります．

1) 送信できたが，待ちタスクがいた場合は，待ちタスクを待ち状態から解除し，データ・キューのデータを渡す
2) 送信できたが，待ちタスクがいない場合は，データ・キューのバッファ内にデータを保管する

データ・キューへの送信(タイム・アウトあり)

C言語API
ER tsnd_dtq(ID dtqid, VP_INT data, TMO tmout);
返り値
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), (E_CTX), (E_MACV), (E_OACV), (E_NOMEM), E_ID, E_NOEXS, E_PAR, E_RLWAI, E_TMOUT, E_DLT

指定したデータ・キューにデータを送信します．tsnd_dtqは，指定したデータ・キューにデータを送信することは同じですが，バッファに保管できなかった場合，一定時間，送信待ち状態に入る部分に特徴があります．送信できた場合は，snd_dtqと同じふるまいになります．

1) 送信できたが，待ちタスクがいた場合は，待ちタスクを待ち状態から解除し，データ・キューのデータを渡す
2) 送信できたが，待ちタスクがいない場合は，データ・キューのバッファ内にデータを保管する

なお，タイム・アウトの指定にポーリング(TMO_POL)を指定すると，psnd_dtqと同じふるまいになり，TMO_FEVRを指定するとsnd_dtqと同じふるまいになります．

データ・キューへの強制送信

C言語API
ER fsnd_dtq(ID dtqid, VP_INT data); ER ifsnd_dtq(ID dtqid, VP_INT data);
返り値
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), (E_CTX), (E_MACV), (E_OACV), (E_NOMEM), E_ID, E_NOEXS, E_ILUSE

指定したデータ・キューにデータを送信します．fsnd_dtq, ifsnd_dtqは，指定したデータ・キューにデータを送信することは同じですが，バッファに保管できなかった場合には，バッファの最後の位置に強制的に上書きします．このため，fsnd_dtq, ifsnd_dtqを使用する場合はデータの消失に関して考慮しておく必要があります．

たとえば，どのような用途で使用するかに関してですが，定期的に送られてくる座標情報や，プロトコルのあるシリアル通信のデータなどで，本来は途中が抜けてほしくはないが，ある条件下ではデータの消失があっても問題ない場合などは，最後のデータが重要であったりします．

ほかにも，2種類のデータがあり，片方がデータの消失を許さない場合など，どうしてもバッファに入れ，送信相手側のタスクに伝えたい場合などは，最後の重要なデータで上書きすることを考える場合があります．

上記のような場合，データの消失を考慮しながらfsnd_dtq, ifsnd_dtqが使用できないかを考慮することになります．

ただし，このような場合でも，バッファの空きがない状態で2回以上呼んだ場合は，上書きしたデータを再度上書きしてしまうので，この部分も考慮が必要です．送信できた場合に関しては，snd_dtqと同じふるまいになります．

先に，データ・キューを使用した同期方法として，データ・キューのバッファ・サイズを0にする方法を説明しましたが，fsnd_dtq, ifsnd_dtqでは最後データ・キューの末尾に上書きするため，データ・キューのバッファ・サイズを0にすることはできません．この場合，返り値としてサービス・コール不正使用(E_ILUSE)が戻ります．

データ・キューからの受信

C言語API
ER rcv_dtq(ID dtqid, VP_INT *p_data);
返り値
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), (E_CTX), (E_MACV), (E_OACV), (E_NOMEM), E_ID, E_NOEXS, E_PAR, E_RLWAI, E_DLT

データ・キューからデータを取り出します．データ・キューからの受信も複数ありますが，違いは受信できなかった場合のふるまいに違いがあります．

rcv_dtqでは受信できなかった場合は，データ・キューへの受信待ち行列に自タスクをつなげます．受信待ち行列はFIFO順になり，要求があった順に処理されます．

データ・キューからの受信(ポーリング)

C言語API
ER prcv_dtq(ID dtqid, VP_INT *p_data);
返り値
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), (E_CTX), (E_MACV), (E_OACV), (E_NOMEM), E_ID, E_NOEXS, E_PAR, E_TMOUT

データ・キューからデータを取り出します．rcv_dtqで受信できなかった場合は，すぐにポーリング失敗(E_TMOUT)を返り値として呼び出し元に返ります．受信待ち状態に入ることはありません．

データ・キューからの受信(ポーリング)

C言語API
ER trcv_dtq(ID dtqid, VP_INT *p_data, TMO tmout);
返り値
ER E_OK(正常終了)またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), (E_CTX), (E_MACV), (E_OACV), (E_NOMEM), E_ID, E_NOEXS, E_PAR, E_RLWAI, E_TMOUT, E_DLT

データ・キューからデータを取り出します．trcv_dtqでは受信できなかった場合は，タイム・アウトの指定を取ります．タイムアウトした場合はタイム・アウト(E_TMOUT)を返り値として呼び出し元に返ります．

なお，tsnd_dtqと同じく，タイム・アウトの指定にポーリング(TMO_POL)を指定すると，prcv_dtqと同じふるまいになり，TMO_FEVRを指定するとrcv_dtqと同じふるまいになります．

● データ・キューの処理速度

割り込み処理では，処理時間が短ければ短いほど，ほかの割り込み処理に影響を与えにくくなるため，良いとされています．

たとえば，割り込み処理は，途中で優先度の高い割り込みへの割り込み許可や，ネストを許さない限り，先に処理を始めたほうが優先されます．そのため，ほかの割り込み処理の割り込み遅延時間が，割り込みの処理時間分だけ長くなります．

シリアルの割り込み処理ではデータをリング・バッファで扱うことが多いのですが，データ・キューのリング・バッファを使用した場合ではどうでしょうか？

処理時間の差を考えるため，バッファ処理の内容を考えてみると，同じリング・バッファを使用しているデータ・キューと比較した場合，データ・キューのほうが処理自体，少し多いようです．

また，アプリケーションの処理に近い所を見ても，シリアルの処理では8ビット長のデータ単位を使用することが多いのですが，データ・キューはワード単位でデータを扱います．シリアル処理の固有データ8ビット分以外に回線の状態などを保存できたとしても，データごとに保存すべきものでもないようです．

以上のことから，シリアル通信などに使う場合は，少しオーバ・ヘッドがかかることを考慮して利用する必要があります．

おわりに

ここまでで同期・通信機能のサービス・コールの一部を説明し終わったことになります．

次回で同期・通信機能を説明し終えることになり，その後は，メモリ・プール管理機能や時間管理機能，時間管理機能の説明を行います．

きしだ・まさみ　(株)フルノシステムズ

TMS320C6713搭載DSPスタータ・キットを使った

C++による DSPオブジェクト指向プログラミング

第4回 FFTクラスの作成

◆三上 直樹

今回は，次回以降のプログラムで使うためのFFTクラスと，このクラスを作るために用いる複素数クラスおよびいくつかの算術演算用関数を作成します．

ディジタル信号処理では，FFT(fast Fourier transform：高速フーリエ変換)がよく使われます．FFTとは，信号のスペクトルを計算機で計算する際に行う演算(これを離散的フーリエ変換という)の演算量を減らして高速に実行するためのアルゴリズムです．このFFTはスペクトルの計算だけでなく，ほかにもいろいろな使い道があります．ディジタル・フィルタを実現するときの計算を高速化するために使うという用途もその一つです．

ところで，このFFTを実行するためには複素数演算が必要になります．もちろん，実数の演算だけでもFFTのプログラムを作ることはできますが，プログラムが長くなり，その結果，バグが紛れ込む可能性も大きくなります．そこで，最初に複素数を扱うためのクラスと，そこで用いるいくつかの算術演算用のプログラムを作ります．このクラスを利用すると，複素数型のデータに対しても"+"や"*"のような演算子を使ってプログラムを書くことができます．そのあとで，FFTの簡単な説明を行い，FFTを実行するためのクラスを作成し，さらにこのクラスを使った簡単なプログラムの例を示します．

1 複素数クラスとは

先にも述べたように，FFTのプログラムを作成する際には複素数型のデータを使うと便利です．C++では通常，標準テンプレート・ライブラリ(STL)が提供されており，このライブラリには複素数を扱うためのテンプレート・クラス(template class)が含まれています．しかし，残念ながらCode Composer Studio(以下，CCS)に付属するC++コンパイラは，STLをサポートしていません．そこで，最初に複素数クラスを作成します．

STLに含まれる複素数を扱うためのクラスは，テンプレート・クラスになっていますが，ここではデータ・メンバを`float`型に限定するので，テンプレート・クラスではなく通常のクラスとします．また，作成するメンバ関数やフレンド関数などはこの連載で使う可能性のあるもののみとします．そのため，STLの複素数クラスに含まれる"`+=`"などの代入演算子や，"`==`"などの関係演算子，"`sin()`"などの多くの算術関数はサポートしていません．

リスト1(`MyComplex.hpp`)に，複素数クラスと複素数を扱ううえで便利な関数をまとめたものを示します．このリストの中でインクルードしている`MyMath.hpp`については第2項で説明します．

● データ・メンバ

複素数の実部(`re`)と虚部(`im`)は，非公開のデータ・メンバで，`float`型とします．

▶ コンストラクタ

最初のコンストラクタはデフォルト・コンストラクタです．その次のコンストラクタはコピー・コンストラクタです．

▶ メンバ関数

メンバ関数としては，複素数の実部を取得する`Real()`と虚部を取得する`Imag()`を用意しました．これら二つの関数は，フレンド関数としても定義しています．そのほか，"`=`"演算子と単項演算子としての"`-`"をメンバ関数にしています．

▶ 演算子のオーバロード

演算子のオーバロード(overloading)の方法については，コラム「演算子のオーバロード」を参照してください．

`Complex`クラスの中でオーバロードを行った演算子は，"`+`"，"`-`"，"`*`"，"`=`"です．なお，"`-`"は，単項演算子および二項演算子として定義しています．表1には定義した演算子の一覧を示します．

割り算を行うための演算子"`/`"は定義していません．その理由は，C6000シリーズのDSPはほかの大部分のDSPと同様に

表1 `Complex`クラスで定義されている演算子の一覧

演算子	演算子のタイプ	説明
+	二項	加算
-	単項	負符号
-	二項	減算
*	二項	乗算
=	二項	代入

リスト 1
複素数クラスとそれに関連する関数
(MyComplex.hpp)

```
//-----------------------------------------------------------------------------
//      複素数クラス
//-----------------------------------------------------------------------------
#ifndef MK_MyComplex

#include <cmath>
#include "MyMath.hpp"
using namespace std;

class Complex
{
private:
    float re, im;
public:
    Complex(const float x = 0.0, const float y = 0.0)
    : re(x), im(y) {}                                                   // デフォルト・コンストラクタ
    Complex(const Complex &z) : re(z.re), im(z.im) {}                   // コピー・コンストラクタ
    float Real() const { return re; }                                   // 実部を取り出す
    float Imag() const { return im; }                                   // 虚部を取り出す
    inline Complex &operator=(const Complex &x);                        // '=', 右辺値が Complex
    inline Complex &operator=(const float x);                           // '=', 右辺値が float
    Complex operator- () const {return (Complex(-re, -im));}            // 単項演算子 '-'

    friend float Real(const Complex &x) { return x.re; }                // 実部を取り出す
    friend float Imag(const Complex &x) { return x.im; }                // 虚部を取り出す

// 加算: '+'
    friend Complex operator+(const Complex &x, const Complex &y)        // Complex + Complex
        { return (Complex(x.re + y.re, x.im + y.im)); }
    friend Complex operator+(const Complex &x, const float y)           // Complex + float
        { return (Complex(x.re + y, x.im)); }
    friend Complex operator+(const float x, const Complex &y)           // float + Complex
        { return (Complex(x + y.re, y.im)); }

// 減算: '-'
    friend Complex operator-(const Complex &x, const Complex &y)        // Complex - Complex
        { return (Complex(x.re - y.re, x.im - y.im)); }
    friend Complex operator-(const Complex &x, const float y)           // Complex - float
        { return (Complex(x.re - y, x.im)); }
    friend Complex operator-(const float x, const Complex &y)           // float - Complex
        { return (Complex(x - y.re, -y.im)); }

// 乗算: '*'
    friend Complex operator*(const Complex &x, const Complex &y)        // Complex * Complex
        { return (Complex(x.re*y.re - x.im*y.im, x.re*y.im + x.im*y.re)); }
    friend Complex operator*(const Complex &x, const float y)           // Complex * float
        { return (Complex(x.re*y, x.im*y)); }
    friend Complex operator*(const float x, const Complex &y)           // float * Complex
        { return (Complex(x*y.re, x*y.im)); }

// 複素数の絶対値の2乗, 偏角, 複素共役, 逆数
    friend float Norm(const Complex &x)                                 // 複素数の絶対値の2乗
        { return (x.re*x.re + x.im*x.im); }
    friend float Arg(const Complex &x)                                  // 複素数の偏角
        { return (atan2( x.im, x.re)); }
    friend Complex Conj(const Complex &x)                               // 複素共役
        { return (Complex(x.re, -x.im)); }
    inline friend Complex Rcp(const Complex &x);                        // 複素数の逆数
};

// 右辺値が Complex の場合の '=' 演算子
inline Complex &Complex::operator=(const Complex &x)
{
    re = x.re;
    im = x.im;
    return *this;
}

// 右辺値が float の場合の '=' 演算子
inline Complex &Complex::operator=(const float x)
{
    re = x;
    im = 0.0;
    return *this;
}

// 複素数の逆数
inline Complex Rcp(const Complex &x)
{
    float tmp = fInv(Norm( x));
    return (Complex(x.re*tmp, -x.im*tmp));
}

// 複素数の絶対値と exp( jx)
inline float Abs(const Complex &x)  { return fSqrt(Norm(x)); }          // 複素数の絶対値
inline Complex Expj(const float x)                                      // exp( jx) の値
        { return (Complex(cosf(x), sinf(x))); }

#define MK_MyComplex
#endif
```

ハードウェアの除算器をもっていないため，割り算を行うと処理時間が長くなってしまうからです．そこで，複素除算の代わりにC67xx固有の命令を使い，高速に複素数の逆数を求める関数Rcp()を定義しています．

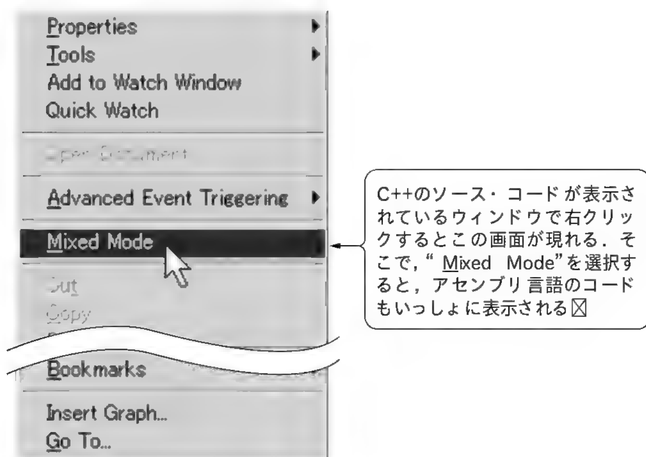


図1　C++のソース・コードの間にアセンブリ言語のコードを表示させるための設定

演算子のなかで，二項演算子として使用する“+”，“-”，“*”はすべてフレンド関数として定義しました[注1]．

ところで，リスト1からわかるように，これらの二項演算子のためにそれぞれ3種類の定義を行っていますが，かならずしも3種類の定義が必要なわけではありません．しかし，ここでは生成されたコードの実効効率を上げるために3種類の定義を行っています．

たとえば，“+”を例として考えてみます．“+”演算子に対して，次のように定義したと仮定します．

```
friend Complex operator+(const Complex &x,
                         const Complex &y)
  { return (Complex(x.re + y.re,
                    x.im + y.im)); }
```

この定義だけで，コンパイラは次の三つのタイプの加算に対して正しく解釈し，正しいコードを生成することはできます．

① Complex型 + Complex型
② Complex型 + float型

注1：これらの二項演算子は，演算子の左側にくる要素がComplex型のオブジェクトの場合にはメンバ関数としても定義することができる．その方法については，コラム「演算子のオーバロード」を参照．

Column 1

演算子のオーバロード

演算子を多重定義，つまりオーバロードする場合は，演算子関数と呼ばれる関数を作ります．多くの場合，この演算子関数はクラスのメンバ関数またはフレンド関数として記述します．

● 二項演算子のオーバロード（その1：メンバ関数として定義する場合）

たとえば，次のような文があるとします．

```
c = a + b;
```

このとき，加算の記号“+”が二項演算子です．メンバ関数で二項演算子をオーバロードする場合，その関数は仮引き数を一つだけもつような関数として記述しなければなりません．たとえば，Complexクラスが次のように定義されているものとして説明します[注A]．

```
class Complex
{
private:
  float re, im;
public:
  Complex(const float x = 0.0,const float y = 0.0)
    : re(x), im(y) {}
  .....
  float Real() const { return re; }
  float Imag() const { return im; }
  .....
  Complex operator+(const Complex &x) const
    { return (Complex(re + x.re, im + x.im)); }
  .....
}
```

注A：説明に必要な部分のみを書いている．なお，説明のつごうのため，リスト1の内容とは多少異なっている．

このとき，a, b, cがComplex型のオブジェクトであるとすると，

```
c = a + b;  // a, b, c: Complex
```

は次のように解釈されます．

```
c = a.operator+(b);
```

つまり，“operator+”をComplexクラスのメンバ関数名と考え，演算子“+”の右側の要素を，このメンバ関数の引き数とみなせばよいということになります．したがって，Complex型のオブジェクトであるaに対して，“operator+”という名前のメンバ関数を作用させるというぐあいに解釈することができます．以上のことから，演算子“+”の左側の要素aはComplex型のオブジェクトである必要があります．

それでは，

```
c = a + b;  //a, c: Complex; b: float
```

で，a, cがComplex型のオブジェクトでbがfloat型の場合はどうでしょうか．この場合は次のように解釈されます．

```
c = a.operator+(Complex(b));
```

つまり，float型のデータをいったんComplex型のオブジェクトに変換するという処理が付け加えられるため，実行効率が悪くなります．そこで，それを避けるためには，次のメンバ関数を付け加える必要があります．

③ float型 + Complex型

しかし，②と③のケースでは，float型のデータを，Complexクラスのコンストラクタを使ってComplexオブジェクトに変換するという処理が付け加わったコードが生成されるため，効率が悪くなります．そこで，②の場合に対しては，

```
friend Complex operator+(const Complex &x,
                         const float y)
  { return (Complex(x.re + y, x.im)); }
```

③の場合に対しては，

```
friend Complex operator+(const float x,
                         const Complex &y)
  { return (Complex(x + y.re, y.im)); }
```

という定義を行っています．

以上のことを確認するために，①と②の場合について，コンパイラが生成したアセンブリ言語のコードを見てみることにします．CCSでは，C++のソース・コードが表示されているウィンドウで，マウスを右クリックすると，図1に示す画面が現れ，“Mixed Mode”を選択すると，C++のソース・コードの間に，対応するアセンブリ言語のコードが表示されます注2．その表示を図2に示します．この図は，

```
x1 = a1 + a2; // すべてComplex型
x2 = a1 + b1; // b1のみfloat型
```

というC++のソース・コードに対応するアセンブリ言語のコードです．ただし，a1，a2，x1，x2はComplex型のオブジェクト，b1はfloat型の変数です．

なお，この図に表示されているアセンブリ言語のコードは“Debug”版で，最適化とデバッグに関するオプションはデフォルトのままでビルドした場合のものです注3．

注2：“Debug”版の場合は，デフォルトでアセンブリ言語のコードが表示される．しかし，“Release”版の場合は，デフォルトでアセンブリ言語のコードは表示されないので，プロジェクトのオプションの設定が必要となる．なお図2では，スペースを節約するため，アセンブリ言語のコードの表示形式は，デフォルトと異なった状態に設定している．デフォルトでは，ここに表示されているもの以外に，アドレスとコードに対する16進数も表示される．

注3：“Debug”版としてビルドした場合であっても，最適化とデバッグに関するオプションを変更した場合には，アセンブリ言語のコードがこれと異なる場合もある．なお，“Release”版としてビルドした場合，デフォルトの設定ではアセンブリ言語のコードは表示されないので，表示させたい場合はデバッグに関するオプションを変更する必要がある．

```
Complex operator+(const float x) const
  { return (Complex(re + x, im)); }
```

● 二項演算子のオーバロード（その2：フレンド関数として定義する場合）

```
c = a + b;  //b, c: Complex; a: float
```

この文で，今度はb，cがComplex型のオブジェクトでaがfloat型の場合は，上で説明した二つのメンバ関数だけではコンパイラが解釈できません．したがって，新たに関数の定義を追加する必要があります．しかし，この場合は先に説明した二つのケースのようにメンバ関数として記述することができません．その理由は演算子“+”の左側の要素aがComplex型のオブジェクトにはなっていないからです．

このような場合は，メンバ関数として記述することはできないので，次のように定義することになります．

```
Complex operator+(const float x, const Complex &y)
  { return(Complex(x+y.Real(), y.Imag())); }
```

基本的には，これでもかまいませんが，メンバ関数を使ってComplexクラスの非公開(private)のデータ・メンバにアクセスするので，効率が悪くなります．そこで，非公開データ・メンバ(re，im)に直接アクセスするようにします．そのためには，この関数を次のようにComplexクラスのフレンド関数にする必要があります．

```
friend Complex operator+(const float x,
                         const Complex &y)
  { return (Complex(x + y.re, y.im)); }
```

この定義により，

```
c = a + b;
```

は，

```
c = operator+(a, b);
```

というぐあいに解釈されます．

ところで，演算子“+”の右側および左側が両者共にComplexクラスのデータ・メンバであっても，フレンド関数で以下のように記述することもできます．

```
friend Complex operator+(const Complex &x,
                         const Complex &y)
  { return (Complex(x.re + y.re,
                    x.im + y.im)); }
```

なお，このコラムでは二項演算子を定義する方法を2通り説明しましたが，リスト1では，二項演算子を定義する際に，すべてフレンド関数として定義しています．

● 単項演算子のオーバロード（メンバ関数として定義する場合）

メンバ関数で単項演算子をオーバロードする場合，その関数は仮引き数をもたないような関数として記述しなければなりません．たとえば，x，yがComplex型のオブジェクトであるものとして，次の文を有効にするためには単項演算子“-”をオーバロードする必要があります．

```
y = -x;
```

そのためには次の定義を付け加える必要があります．

```
Complex operator-() const { return (Complex(-re,
                                            -im)); }
```

この定義により，

```
y = x.operator-();
```

というぐあいに解釈されます．

図2(a)は，

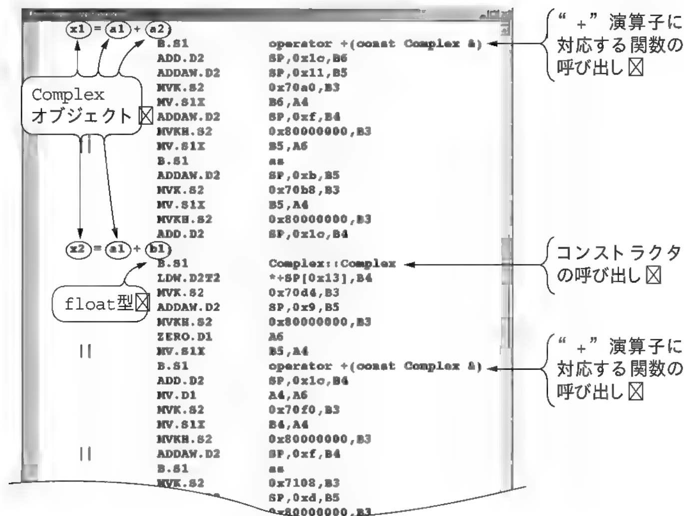
```
friend Complex operator+(const Complex &x,
                         const Complex &y)
```

だけを宣言している場合を示しています．この場合は演算子関数の引き数はどちらもComplex型になっています．

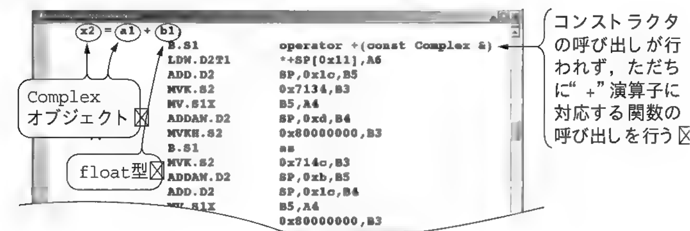
図2(b)は，さらに，

```
friend Complex operator+(const Complex &x,
                         const float y)
```

という宣言も付け加えた場合です．つまり，第2引き数をfloat型として演算子関数を宣言した場合です．なお，"x1=a1+a2;"に対するコードは同じなので，図2(b)では省略しています．



(a) "+"演算子に対する定義がfriend Complex operator+(const Complex &x, const Complex &y)に対応するものだけの場合



(b) "+"演算子に対する定義がfriend Complex operator+(const Complex &x, const float y)に対応するものも追加した場合

図2 "+"演算子の定義の違いに対する，生成されるアセンブリ言語のコードの違い

図2(a)の場合，"x2=a1+b1;"に対するアセンブリ言語のコードを見ると，Complexクラスのコンストラクタを使って，float型の変数であるb1をいったんComplex型のオブジェクトに変換した後に，"+"演算子に対する処理を行っています．一方，図2(b)では，float型の変数であるb1をいったんComplex型のオブジェクトに変換するというコードは生成されていないことがわかります．その結果，実行効率が高くなります．

以上で説明してきたことは，"-"，"*"の場合でも同様です．

なお，引き数がComplex型のオブジェクトの場合に，値渡しではなく参照渡しになっているのは実行効率を上げるためです．ポインタ渡しでもかまわないのですが，ポインタ記法("*"や"->"など)を使うため，ソース・プログラムの記述が煩雑になってしまいます．

"="と単項演算子"-"は，メンバ関数として定義しました．

"="演算子は，右辺がComplex型のオブジェクトの場合とfloat型の場合の二つを定義しました．"="演算子に対してfloat型に対する定義を行ったのは，二項演算子"+"での説明と同じ理由です．

● そのほかのフレンド関数と非フレンド関数

演算子のオーバロードで使っている以外のフレンド関数としては，絶対値の2乗を求めるNorm()，偏角を求めるArg()，複素共役を求めるConj()，逆数を求めるRcp()を定義しています．また，複素数の実部および虚部を取得する関数は，メンバ関数としての定義のほかに，フレンド関数としても定義しています．

そのほか，絶対値を求める関数Abs()と $\exp(jx)$[注4]を計算

表2 **Complex**クラスで定義されている関数およびそれと関連する関数の一覧

関　数	関数の種類	戻り値	戻り値の型
Complex(const float x, const float y)	メンバ関数(コンストラクタ)	xを実部，yを虚部とするComplex型のデータ	Complex型
Real()	メンバ関数	Complex型の実部	float型
Imag()	メンバ関数	Complex型の虚部	float型
Real(const Complex &x)	フレンド関数	Complex型の実部	float型
Imag(const Complex &x)	フレンド関数	Complex型の虚部	float型
Norm(const Complex &x)	フレンド関数	Complex型の絶対値の2乗	float型
Arg(const Complex &x)	フレンド関数	Complex型の偏角	float型
Conj(const Complex &x)	フレンド関数	Complex型の複素共役数	Complex型
Rcp(const Complex &x)	フレンド関数	Complex型の逆数	Complex型
Abs(const Complex &x)	一般の関数†	Complex型の絶対値	float型
Expj(const float x)	一般の関数†	$\exp(jx)$の値，j：虚数単位	Complex型

†：メンバ関数でもフレンド関数でもない通常のグローバル関数の意味．

する関数Expj()は，Complexクラスの非公開データ・メンバにアクセスする必要はないので，フレンド関数ではなく通常のグローバル関数として記述しています．

表2には複素数クラスで定義されたメンバ関数，フレンド関数，非フレンド関数の一覧を示します．

なお，関数Rcp()のなかで使っているfInv()，および関数Abs()のなかで使っているfSqrt()は，インクルード・ファイルMyMath.hppのなかで定義されている関数で，第2項で説明します．

2 複素数クラスで使用している関数

Complexクラスを定義するときに使っている関数のなかで，C67xxの固有の命令を利用すると計算の高速化が可能のものがあります．そこで，それらをまとめたものをリスト2(MyMath.hpp)に示します．また，表3にその一覧を示します．

C67xxは，逆数の近似値および平方根の逆数の近似値を1マシン・サイクルで求める命令を持ち，C/C++コンパイラはこの命令を直接使うためのintrinsics関数をサポートしています．この二つのintrinsics関数の名前は次のようになっています．

逆数の近似値………………… _rcpsp()

平方根の逆数の近似値……… _rsqrsp()

これらのintrinsics関数は，ニュートン・ラフソン(Newton-Raphson，注：発音は原音読みとした)法で逆数や平方根を求める際の初期値を求めるために使います．

● ニュートン・ラフソン法

ニュートン・ラフソン法は，方程式の根を繰り返しにより求める方法の一つです．根を求めたい方程式を$f(x)=0$とすると，ニュートン・ラフソン法では以下の反復を行い，根の近似値を真の値に近づけて行きます．

$$x^{(n+1)}=x^{(n)}-\frac{f(x^{(n)})}{f'(x^{(n)})},\quad n=0,1,2,\cdots \quad\cdots\cdots(1)$$

ここで，$x^{(0)}$は初期値，$x^{(n)}$は反復をn回行ったときの根の近似値，$f'(x^{(n)})$は$x=x^{(n)}$における$f(x)$の微分係数です．

ニュートン・ラフソン法では，根の近似値が真の値に十分近い値の場合，正しい桁数は1回の反復ごとに約2倍に増えることが知られています．一方，intrinsics関数の_rcpsp()，_rsqrsp()を使うと仮数部で8ビットの精度が得られます．したがって，2回の繰り返を行うと，仮数部の精度が32ビット程度[注5]になるので，float型の精度(仮数部：24ビット)を得るためには2回の繰り返しで十分です．

● 逆数の計算

aの逆数は，次の方程式の解として与えられます．

$$f(x)=\frac{1}{x}-a=0 \quad\cdots\cdots(2)$$

これをニュートン・ラフソン法で解く場合，反復の式は次のようになります．

$$x^{(n+1)}=x^{(n)}(2-ax^{(n)}) \quad\cdots\cdots(3)$$

これを利用して逆数を求める関数がfInv()です．

● 平方根の計算

aの平方根は，次の方程式の解として与えられます．

$$f(x)=x^2-a=0 \quad\cdots\cdots(4)$$

これをニュートン・ラフソン法で解く場合，反復の式は次のようになります．

$$x^{(n+1)}=0.5\left(x^{(n)}+\frac{a}{x^{(n)}}\right) \quad\cdots\cdots(5)$$

この式のなかには除算が入っています．ところで，DSPは一般にハードウェア除算器を備えていません．C6000シリーズ

リスト2　TMS320C67XX用の算術関数を，Intrinsics関数を利用して高速に計算するためのプログラム(MyMath.hpp)

```
//-----------------------------------------------------------
// Intrinsics 関数を利用する TMS320C67xx 用の算術関数
// fInv, fInvSqrt, fSqrt
//-----------------------------------------------------------
#ifndef MK_MyMath

#include <cassert>

inline float fInv(const float x);
inline float fInvSqrt(const float x);
inline float fSqrt(const float x);

// 逆数
inline float fInv(const float x)
{
    float y = _rcpsp(x);    // 1/x の近似値
    for (int i=0; i<2; i++)
        y = y*(2.0f - x*y);
    return (y);
}

// 平方根の逆数
inline float fInvSqrt(const float x)
{
    float y = _rsqrsp(x);   // 1/x の平方根の近似値
    for (int i=0; i<2; i++)
        y = y*(1.5f - 0.5f*x*y*y);
    return (y);
}

// 平方根
inline float fSqrt(const float x)
{
    float y = (x != 0.0f) ? fInvSqrt(x) : 0.0f;
    return (x*y);
}

#define MK_MyMath
#endif
```

表3　Intrinsics関数を利用して高速に実行可能な算術関数の一覧

関　数	戻り値	戻り値の型
fInv(const float x)	xの逆数	float型
fInvSqrt(const float x)	xの逆数の平方根	float型
fSqrt(const float x)	xの平方根	float型

注4：$\exp(jx)=\cos x+j\sin x$

注5：もちろん，仮数部の精度が32ビットより十分高くなるようにして計算を行った場合．

のDSPもその例外ではありません．したがって，そのようなDSPで除算を行うと実行時間が長くなり，望ましくありません．そこで，一度 a の平方根の逆数を求めてから a の平方根を求めるという方法を使います．

a の平方根の逆数（$1/\sqrt{a}$）は，次の方程式の解として得られます．

$$f(x) = \frac{1}{x^2} - a = 0 \quad \cdots\cdots (6)$$

これをニュートン・ラフソン法で解く場合，反復の式は次のようになります．

$$x^{(n+1)} = 0.5x^{(n)}\left\{3 - a\left(x^{(n)}\right)^2\right\} \quad \cdots\cdots (7)$$

この反復式には除算が出てこないので，まずこれを使った関数 `fInvSqrt()` のプログラムを作ります．

$1/\sqrt{a}$ が求められれば，$a(1/\sqrt{a})$ で a の平方根を求めることができます．そのようにして作った平方根を求める関数が `fSqrt()` です．

3 DFTとFFTのアルゴリズム

FFTとは，DFT（discrete Fourier transform：離散的フーリエ変換）の計算スピードを上げるためのアルゴリズムです．そこで，はじめにDFTについて簡単に説明してからFFTアルゴリズムの説明を行います．

● DFT

▶ DFTの定義

標本化された信号を $g[n]$ とすると，そのDFTである $G[k]$ は次の式で計算されます．

$$G[k] = \sum_{n=0}^{N-1} g[n]\exp\left(\frac{-j2\pi nk}{N}\right), \quad k = 0, 1, \cdots, N-1 \quad \cdots (8)$$

リスト3　DFTの定義を直接使って計算するプログラム（DFT.cpp）

```
//----------------------------------------------------------
// 複素データのDFTを計算する
//----------------------------------------------------------

#ifndef MK_DFT

#include "MyComplex.hpp"

void DFT(const Complex x[], Complex y[], int nDFT);

// DFTの定義を直接使うDFTの計算
void DFT(const Complex x[], Complex y[], int nDFT)
{
    float PI2ovN = 6.28318531f*fInv(nDFT);

    for (int k=0; k<nDFT; k++)
    {
        y[k] = 0.0;
        for (int n=0; n<nDFT; n++)
            y[k] = y[k] + x[n]*Expj(-PI2ovN*n*k);
    }
}

#define MK_DFT
#endif
```

ここで，j は虚数単位で，$j = \sqrt{-1}$ です．また，N はDFTの計算に用いるデータ数，つまり標本化された信号の数です．

これとは逆に，ある信号のDFTを $G[k]$ とすると，この $G[k]$ が与えられたときに，元の信号 $g[n]$ を次の式で計算することができます．

$$g[n] = \frac{1}{N}\sum_{k=0}^{N-1} G[k]\exp\left(\frac{j2\pi nk}{N}\right), \quad n = 0, 1, \cdots, N-1 \quad \cdots (9)$$

この操作は，逆DFT（IDFT）と呼ばれています．あるデータのDFTを計算し，その逆DFTを計算すると元のデータに戻ることになります．

これらの式で，表記を簡潔にするため，複素指数関数の部分を次のように置き換えて表現する場合もあります．

$$W_N = \exp\left(\frac{-j2\pi}{N}\right) \quad \cdots\cdots (10)$$

この W_N は回転因子，またはひねり係数（twiddle factor）などと呼ばれています．これを使うと，式（8）は次のように書くことができます．

$$G[k] = \sum_{n=0}^{N-1} g[n]\,W_N^{nk}, \quad k = 0, 1, \cdots, N-1 \quad \cdots\cdots (11)$$

この W_N という表現は，FFTのところでも使います．

▶ DFTのプログラム

実用的な場面では，DFTを式（8）に基づいて直接計算するということはあまり行いません．しかし，後に出てくるFFTのプログラムの結果を確かめる場合に，比較の対象として使えるようにDFTのプログラムを作成しました．これをリスト3（DFT.cpp）に示します．

● FFTアルゴリズム

FFTアルゴリズムを使うとDFTの計算量を大幅に削減できるため，計算を高速に実行することができます．したがって，DFTを応用する際には，通常FFTがよく使われます．私たちがもっともよく使うFFTは，データ数が 2^M（M：整数）になるときに適用できるもので，これを“2を基底とする（radix-2）アルゴリズム”といいます．また，FFTのアルゴリズムにはいろいろなものがありますが，ここでは周波数間引き（decimation-in-frequency）アルゴリズムについて説明します．

図3に，$N = 2^4 = 16$ に対する“2を基底とする周波数間引きFFT”を示します．この図から，このFFTは図4に示す基本演算の組み合わせで構成されていることがわかります．この基本演算はバタフライ演算（butterfly operation）と呼ばれています．

次に，処理の手順について説明します．第1段目ではデータを前半と後半の二つのグループに分け，$N/2 = 8$ だけ離れたところにあるデータどうしでバタフライ演算を行います．また，バタフライ演算で使われている回転因子 W_{16}^k の k は上から順に0, 1, 2, 3, 4, 5, 6, 7になっています．

第2段目では，まず第1段目で分けられた二つのグループを，さらにそれぞれ二つのグループに分け，全体では四つのグループに分けます．次に，$N/4 = 4$ だけ離れたところにあるデータ

第1段目⊠　第2段目⊠　第3段目⊠　第4段目⊠

$W_{16}{}^{k} = \cos\left(\frac{2\pi ⊠}{16}\right) - j\sin\left(\frac{2\pi ⊠}{16}\right)$

図3　周波数間引きによるFFTアルゴリズム(N=16の場合)

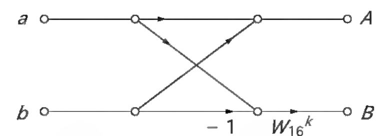
どうしでバタフライ演算を行います．また，バタフライ演算で使われている回転因子 W_{16}^{k} の k は上から順に 0,2,4,6,0,2,4,6 になっています．

同様に，第3段目，第4段目も同じように処理を行っていくと，最後の段の出力に DFT の結果が現れます．

この例では，全体は4段で構成されていますが，データ数が 2^M のときは，全体が M 段で構成されることになります．

このFFTを使う場合に，図3で出力側のデータの並びかたに注意する必要があります．入力データ $g[n]$の並びかたは，[　]の中の数字の小さいものから大きいものへと順に並んでいるので，何も注意する必要はありません．しかし，求められた $G[k]$の並びかたは，[　]の中の数字の小さいものから順にはなっていません．このような並びかたをビット逆順(bit reversal)の並びかたと呼んでいます．したがって，このアルゴリズムを使う場合は，ビット逆順の順番に並んでいるデータ $G[k]$を普通の順番に並べ換える必要があります．

ビット逆順の数は次のようにして作ることができます．ある



A=⊠+⊠

B=(⊠− ⊠) ⊠$_{16}{}^{k}$

$W_{16}{}^{k} = \exp\left(\frac{-j2\pi ⊠}{16}\right)$

$= \cos\left(\frac{2\pi ⊠}{16}\right) -$ ⊠$\sin\left(\frac{2\pi ⊠}{16}\right)$

図4　バタフライ演算

数が M ビットの2進数で，次のように表されるものとします．

b_{M-1}， …， b_2， b_1， b_0

ただし，b_m(m=0， 1， …． M− 1)は，0または1とします．このとき，各ビットの順番を入れ換えると次のようになります．

b_0， b_1， b_2， …， b_{M-1}

N=16の場合の通常の順番とビット逆順の順番の対応を表4に示します．

このアルゴリズムで逆FFTを行うためには，二つの簡単な修正を行う必要があります．一つ目の修正は，回転因子を W_N から W_N^{-1} に変更することです．もう一つの修正は，最後の結果に $1/N$（N：データ数）を乗算することです．

4 FFTクラスの作成とFFTプログラム

効率のよいFFTのプログラムを作成するために，次の方法を採用します．式（10）の回転因子やビット逆順の並べ替えのためのデータは，FFTそのものとは切り離し，あらかじめ値を計算して表として配列に格納しておきます．そして，FFTの実行時には，それを配列から読み出すようにします．この表を作成する作業は最初に1度実行すればよいので，この部分はコンストラクタとして記述することができます．そのため，FFTのプログラムは，クラスを使って実現するために適しています．

なお，今回はデータが複素数の場合のFFTプログラムを作成しますが，FFTを使ってフィルタを実現する際は，データが実数なので，実数データに対するFFTを使ったほうが高速になります．そこで，実数データに対するFFTも作成しますが，これは次回にまわします．この両者には共通の部分があるので，その部分は基底クラスBaseFFTとし，この基底クラスを継承して複素数データのFFTと実数データのFFTを作成します．

表4　ビット逆順の表

通　常		ビット逆順	
10進数	2進数	2進数	10進数
0	0000	0000	0
1	0001	1000	8
2	0010	0100	4
3	0011	1100	12
4	0100	0010	2
5	0101	1010	10
6	0110	0110	6
7	0111	1110	14
8	1000	0001	1
9	1001	1001	9
10	1010	0101	5
11	1011	1101	13
12	1100	0011	3
13	1101	1011	11
14	1110	0111	7
15	1111	1111	15

● FFTの基底クラス

リスト4（a）には，FFTの基底クラスBaseFFTのヘッダ・ファイル（MyFFTBase.hpp）を，リスト4（b）にはメンバ関数の定義（MyFFTBase.cpp）を示します．このFFTクラスは，FFTおよび逆FFTに対応しています．

▶ ヘッダ（**MyFFTBase.hpp**）

限定公開データ・メンバのnFFT，n_absおよびnInv[注6]はconstメンバとして宣言されています．そのため値を代入できないので，メンバ初期設定の構文を使って値を設定します[注7]．限定公開メンバの中で宣言されている二つのポインタw_tbl，b_tblは，FFTを行ううえで共通に使用するためにあらかじめ計算されるデータを格納する領域のポインタです．w_tblは，回転因子 W_N^k のためのポインタ，b_tblはビット逆順の並べ替え用データのポインタです．

限定公開部で宣言している二つのメンバ関数Swap()，FFT_Loop()は，いずれもBaseFFTの派生クラスで使うためのものです．メンバ関数Swap()は，二つのComplexオブジェクトを交換するためのインライン関数です．そのほか，FFT実行の際に共通に使うルーチンであるFFT_Loop()をメンバ関数として宣言しています．

公開部では，コンストラクタの宣言を行い，デストラクタはインライン関数として定義しています．

▶ メンバ関数の定義（**MyFFTBase.cpp**）

ここでは，コンストラクタとメンバ関数FFT_Loop()の定義を行っています．

コンストラクタで行っていることは，大きく三つに分けられます．

一つ目は，引き数のチェックです．FFTのデータ数は，引き数として与えられますが，この値を N とすると，次の条件を満足する必要があります．

$|N|>0$ で，かつ $N=2^M$：M は整数

このいずれかが満足されない場合は，関数assert()[注8]が実行箇所に関する情報を出力し，プログラムが異常終了します．

なお，コンストラクタの引き数nが負の場合は，回転因子の計算で，逆FFT（IFFT）に対応する値を計算します．したがって，BaseFFTを使ってFFTとIFFTを行う場合は，データ数が同じであっても，オブジェクトの宣言をそれぞれ別に行う必要があります．

二つ目は，あらかじめ計算しておく表（回転因子とビット逆順）のための領域の確保です．そのため演算子newを使い，ヒープ領域にメモリを確保します．このときにメモリを確保できない場合，本来であれば例外処理（try，catchを使う構文）

注6：nInvは逆FFTを行うときに使う．
注7：リスト4（b）に示すコンストラクタの定義の第1行目で，“：”以下のnFFT(n)，n_abs(abs(n))およびnInv(fInv(abs(n)))によりメンバ初期設定を行っている．
注8：assert()はマクロとして定義されており，カッコ内の式の値が偽（false）のとき，メッセージを出力したあと，関数abort()を呼び出して異常終了する．

リスト 4　周波数間引き FFTのための基底クラス

```
//--------------------------------------------------------
// FFT クラス( 周波数間引きアルゴリズム)
// © 三上直樹, 2004 年
//--------------------------------------------------------
#ifndef MK_MyFFTBase

#include <cassert>     // assert() で使用
#include <new>         // new, delete, set_new_handler() で使用
#include "MyComplex.hpp"
using namespace std;

//----------------------
// FFT の基底クラス
//----------------------
class BaseFFT
{
protected:
    const int nFFT, n_abs;
    const float nInv;  // FFT のデータ数の逆数
    int n_half, next;
    Complex  xtmp;
    Complex  *w_tbl;
    int *b_tbl;
    void Swap(Complex &a, Complex &b)
            { Complex tmp = a; a = b; b = tmp; }
    void FFT_Loop(Complex x[]);
public:
    BaseFFT(int n);    // コンストラクタ
    ~BaseFFT() { delete[] w_tbl; delete[] b_tbl; }  // デストラクタ
};

#define MK_MyFFTBase
#endif
```

(a) MyFFTBase.hpp

```
//--------------------------------------------------------
// FFT クラス( 周波数間引きアルゴリズム)
// © 三上直樹, 2004 年
//--------------------------------------------------------
#include "MyFFTBase.hpp"

//----------------------
// FFT の基底クラス
//----------------------

// uFFT のコンストラクタ
// n > 0 の場合: FFT
// n < 0 の場合: IFFT
BaseFFT::BaseFFT(int n) : nFFT(n), n_abs(abs(n)),
                                        nInv(fInv(abs(n)))
{
    int   pw2, n_abs2;
    float arg;

    set_new_handler( 0);
    assert(nFFT);       // データ数=0 の場合に異常終了
    pw2 = n_abs;
    while ((pw2 & 0x1) == 0) pw2 = pw2 >> 1;
    assert(pw2 == 0x1);// データ数が 2 のべき乗の数ではない場合に異常終了
    n_abs2 = n_abs >> 1;
    w_tbl = new Complex[n_abs2];
    assert(w_tbl);      // 領域確保に失敗した場合に異常終了
    b_tbl = new int[n_abs];
    assert(b_tbl);      // 領域確保に失敗した場合に異常終了

// 回転因子の表を作成
    arg = 6.28318531f*fInv(nFFT);
    for (int m=0; m<n_abs2 ;m++) w_tbl[m] = Expj(-arg*m);

// ビット逆順の並べ替えを行うための表を作成
    n_half = n_abs2;
    b_tbl[0] = 0;
    for (int m=1; m<n_abs; m=m<<1)
    {
        for (int k=0; k<m; k++) b_tbl[k+m] = b_tbl[k] + n_half;
        n_half = n_half >> 1;
    }
}

// 周波数間引きアルゴリズムで使用するルーチン
void BaseFFT::FFT_Loop(Complex x[])
{
    int kx, n_half2;

    n_half2 = n_half << 1;
    for (int kp=0; kp<n_abs; kp=kp+n_half2)
    {
        kx = 0;
        for ( int k=kp; k<kp+n_half; k++)
        {
            xtmp = x[k+n_half];
            x[k+n_half] = ( x[k] - xtmp)*w_tbl[kx];
                                            // バタフライ演算
            x[k] = x[k] + xtmp;            // バタフライ演算
            kx = kx + next;
        }
    }
    n_half = n_half >> 1;
}
```

(b) MyFFTBase.cpp

Column 2　set_new_handler() について

new 演算子で領域確保を失敗した場合の動作は，C++ が最初に開発されたときから何度か変更されています．最新の C++(標準 C++ : ISO/IEC 14882, Standard for C++ Programming Language)では基本的に例外処理(try，catch を使う構文)で対処するような言語仕様になっています．

ところが，CCS のコンパイラは現在のところ例外処理をサポートしておらず，new 演算子で領域確保を失敗した場合に，デフォルトではただちにプログラムを異常終了するようなコードを生成します．その結果，デバッグする際にどこで領域確保に失敗したのか調べようとしても，その場所がわからなくなります．そこで，領域確保に失敗した場所がわかるようにします．そのためには new 演算子で領域確保を失敗した場合にデフォルトの動作とは別の動作をさせる必要があります．

そのために使うのが関数 set_new_handler()です．new 演算子を使う前に，関数 set_new_handler()を使って必要な設定を行っておくと，それに応じた動作を行います．そこで，関数 set_new_handler()の使いかたを簡単に紹介します．

まず，関数 set_new_handler()を使うためには，次のインクルード文が必要になります．

```
#include <new>
```

関数 set_new_handler()を使った設定には，以下に示すように 2通りの引き数の与えかたがあり，それぞれについて new 演算子で領域確保に失敗した場合の動作は異なります．

● 引き数に 0を与えた場合

領域確保に失敗した場合に NULL ポインタが返されます．リスト 4ではこの方法を採用しています．

● 引き数に関数名を与えた場合

引き数に，すでに宣言されている関数名を与えると，領域確保に失敗した場合にその関数が実行されます．

を使うべきですが，C6000シリーズ用のコンパイラは現在のところこの機能をサポートしていません[注9]．そこで，このプログラムでは関数`set_new_handler()`[注10]と`assert()`を使い，領域が確保できない場合に，メッセージを出して異常終了するようにしています．

三つ目は，回転因子とビット逆順のためのデータの計算を行います．回転因子の計算で使っている関数`Expj()`は，リスト1(`MyComplex.hpp`)のなかで定義されている関数で，引き数として与えられた値をxとすると，$\cos x+j\sin x$の値を計算します．ビット逆順並べ替え用データの求めかたは，図5に示します．

メンバ関数`FFT_Loop()`は，図3の一つの段に相当する処理を行います．

● FFTの派生クラス(データが複素数の場合)

リスト5(a)には`BaseFFT`の派生クラスである`ComplexFFT`のヘッダ・ファイル(`MyFFTComplex.hpp`)を，リスト5(b)にはメンバ関数の定義(`MyFFTComplex.cpp`)を示します．

ヘッダで定義されているコンストラクタは，基底クラスにFFTのデータ数を渡し，基底クラスのコンストラクタを起動するだけです．

`MyFFTComplex.cpp`では，基底クラスのメンバ関数`FFT_Loop()`を使って，FFTを実行します．処理の手順の中で，最終段の処理は図3に示すものとは多少異なっています．図3の最終段(第4段目)のバタフライ演算ではW_{16}^{0}の乗算が行われます．ところで，

$$W_{16}^{0}=1$$

なので，最終段では乗算が不要になります[注11]．そこで，プログラムではこの部分だけ全体のループ処理から除外して，バタフライ演算の計算を次のように変更しています．

$$\begin{matrix} A=a+b \\ B=(a-b)W_{16}^{0} \end{matrix} \Rightarrow \begin{matrix} A=a+b \\ B=a-b \end{matrix}$$

FFTの処理が終了したら，次にビット逆順の並べ替えの操作が行われます．最後に，コンストラクタの引き数`n`が負の場合は逆FFTを行うので，得られた結果に$1/N$(N：データ数)を乗算します[注12]．

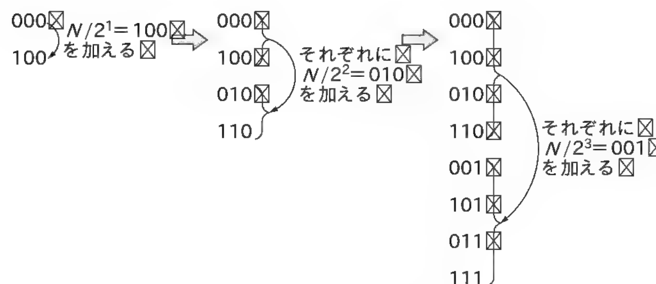


図5　ビット逆順の表の作りかた(N=8の場合)

● FFTクラスのライブラリ作成

ライブラリの作成方法については，本連載第2回目ですでに説明したので，ここでは簡単に示します．

まず，作成するライブラリは"Release"版にするので，ツール・バーの"Debag"と表示されたドロップダウン・コンボ・ボックスで"Release"を選択します．次に，必要なソース・プログラムをプロジェクトに追加します．図6には"Project view"ウィンドウの"Source"へソース・ファイルを追加したようすを示します[注13]．アーカイバのオプション設定では，作

リスト5　複素信号のFFTのための派生クラス

```
//-----------------------------------------------------
// BaseFFT の派生クラス(データが複素数の場合)
//-----------------------------------------------------
#ifndef MK_MyFFTComplex

#include "MyFFTBase.hpp"

class ComplexFFT : public BaseFFT
{
public:
    ComplexFFT(int n) : BaseFFT(n) {} // コンストラクタ
    void Execute(Complex x[]);        // FFT の実行
};

#define MK_MyComplex
#endif
```

(a) `MyFFTComplex.hpp`

```
//-----------------------------------------------------
// BaseFFT の派生クラス(データが複素数の場合)
//-----------------------------------------------------
#include "MyFFTComplex.hpp"

// 複素データの FFT の実行
void ComplexFFT::Execute(Complex x[])
{
    n_half = n_abs >> 1;
// 第1，第2，‥‥，第 log2(nFFT)-1 段目の処理
    for (next=1; next<(n_abs>>1); next<<=1) FFT_Loop(x);
// 最後の1 段の処理
    for (int k=0; k<n_abs; k=k+2)
    {
        xtmp = x[k+1];
        x[k+1] = x[k] - xtmp;
        x[k] = x[k] + xtmp;
    }

// ビット逆順の並べ替え
    for (int k=0; k<n_abs; k++)
        if (k < b_tbl[k]) Swap(x[k], x[b_tbl[k]]);

// IFFT の場合に以下の処理を行う
    if (nFFT < 0)
        for (int k=0; k<n_abs; k++) x[k] = nInv*x[k];
}
```

(b) `MyFFTComplex.cpp`

注9：筆者の使用しているCCS付属のコンパイラはVersion4.32．
注10：関数`set_new_handler()`は，コラム2の「`set_new_handler()`について」を参照のこと．
注11：実際には，最終段の一つ前の段も，回転因子がすべてW_{16}^{0}かW_{16}^{4}であり，$W_{16}^{4}=j$であるので，乗算が不要になる．このことを利用することで演算量をさらに減らすことができる．
注12：$1/N$を乗算するのは，逆DFTの定義として式(9)を採用しているためである．

成されるライブラリのファイル名を"myFFT.lib"に設定しています．以上の設定が終わったら，メニュー・バーで[Project | Build]を選択するか，ボタンをクリックしてビルドを行います．

作成されたライブラリは，プロジェクトの入ったフォルダのReleaseフォルダにmyFFT.libという名前で保存されます．このライブラリは，以降で使うので，筆者はmyprojetcsの下のフォルダであるlib[注14]に置いています．また，三つのヘッダ・ファイルMyFFTBase.hpp，MyFFTComplex.hpp，MyFFTReal.hpp[注15]は，myprojetcsの下のフォルダであるinclude[注16]へ置いています．

● FFTクラスを使用したプログラムの例

リスト6にFFTクラスを使用したプログラムの例として，FFTが正しく行われていることを確認するためのプログラム(FFT_Test.cpp)を示します．このプログラムは，データとして乱数を用い，そのデータのDFTをリスト3の関数DFT()で計算した値と，リスト5に示すComplexFFTクラスのメンバ関数Execute()で計算した値を比較するというものです．

データとして，たとえばsin関数を使って生成したデータのように，規則性のあるデータを使った場合に，プログラムがまちがっていても，正しい結果が出る可能性があります．そこで，そのようなことを防ぐために，データとして乱数を使っています．この乱数にはuRandというクラスで，0以上1未満の一様乱数を発生させたものを利用しています．このクラスもリスト6に含まれています．



図6
FFT用ライブラリを作るために，プロジェクトに追加したソース・ファイルのようす
MyFFTReal.cppについては，次回に解説する．

リスト6 ComplexFFTクラスのためのプログラム(FFT_Test.cpp)

```
//-------------------------------------------------------------
// CmplexFFT クラスのテスト
//-------------------------------------------------------------

#include <cstdio>
#include "MyFFTComplex.hpp"
#include "DFT.cpp"
using namespace std;

void print(char c[], const Complex x[], int n);

// 一様乱数生成のためのクラスの宣言
class uRand
{
private:
    unsigned rand_seed;
public:
    uRand(unsigned seed) { rand_seed = seed; }
    float frand();
};

//-------------------------------------------------------------
int main()
{
    const int N = 16;
    ComplexFFT cFFT(N), cIFFT(-N);
    uRand v(0);

    Complex x1[N], x2[N], y[N];

    for (int j=0; j<N; j++)
        x1[j] = Complex(v.frand()-0.5f, v.frand()-0.5f);

    for (int j=0; j<N; j++) x2[j] = x1[j];
    print("Original date", x1, N);

// 複素 FFT の実行
    cFFT.Execute(x1);
    print("Result of ComplexFFT", x1, N);

// DFT の実行
    DFT(x2, y, N);
    print("Result of DFT", y, N);

// 複素 IFFT の実行
    cIFFT.Execute(x1);
    print("Result of ComplexFFT (IFFT)", x1, N);
}
//-------------------------------------------------------------

// FFT の結果のプリントアウト
void print(char c[], const Complex x[], int n)
{
    printf("¥n%s¥n", c);
    for (int j=0; j<n; j++)
        printf("%6d, %8.4f + j(%8.4f)¥n", j, Real(x[j]),
                                             Imag(x[j]));
}
//-------------------------------------------------------------

// 一様乱数生成のためのクラスの定義
// 0 ≦ 戻り値 < 1.0
float uRand::frand()
{
    union    // 無名共有体
    {
        unsigned  m;
        float x;
    };

    rand_seed = 1664525*rand_seed+1013904223;
    m = 0x3f800000 | (rand_seed >> 9);
    return (x - 1.0f);
}
//-------------------------------------------------------------
```

注13：追加したソース・ファイルのなかで，MyFFTReal.cppは実数データのためのFFTである．これについては次回に説明する．
注14：このフォルダは，本連載の第2回目に作成したライブラリを置いたフォルダと同じ．
注15："MyFFTReal.hpp"については次回に説明する．
注16：このフォルダは，本連載の第2回目に作成したライブラリのヘッダ・ファイルを置いたフォルダと同じ．

Column 3 C++でクラスを使うメリット

筆者がC++を使うようになった大きなきっかけの一つとして，複素数を使ったプログラムが簡単に書けるという点があります．C++ならば，言語の仕様として最初から提供されている整数型(intなど)や浮動小数点型(float，doubleなど)で使える四則演算の演算子+，-，*，/が複素数型でも使えます．

複素数に対して四則演算の演算子が使えると非常に便利だという例をあげてみます．たとえば，次の式の値を計算するとします．ただし，a_2，a_1，a_0，xは複素数であるとします．

$$a_2x^2 + a_1x + a_0$$

この計算のプログラムを作成する場合，一般的には次のように変形して計算します．

$$a_2x^2 + a_1x + a_0 = (a_2x + a_1)x + a_0$$

C++で複素数クラスを使った場合とCで，このプログラムを書くと次のようになります．

● C++で複素数クラスを使った場合

```
y = (a2*x + a1)*x + a0;
```

※a2，a1，a0，x，yは複素数型の変数

● Cの場合

```
y = cAdd(cMpy(cAdd(cMpy(a2, x), a1), x), a0);
```

※cAdd()：複素数の和を返す関数

※cMpy()：複素数の積を返す関数

※a2，a1，a0，x，yは複素数に対応する構造体

どちらがわかりやすいかは一目瞭然でしょう．C++のように演算子が使えるほうが，Cのように関数を使う場合に比べて，はるかにわかりやすいプログラムになります．

リスト 7 FFTtest.cppのビルドの再に使用したリンカ・コマンド・ファイル(lnk_FFT.cmd)の一部

```
-stack 0x1000
-heap  0x8000
-l rts6700.lib                          /* run-time library for C67xx */
-l csl6713.lib                          /* chip support library       */
-l dsk6713bsl.lib                       /* board support library      */
-l c:¥ti¥myprojects¥lib¥aic23.lib       /* library built by MIKAMI    */
-l c:¥ti¥myprojects¥lib¥myFFT.lib       /* library built by MIKAMI    */

           (以下省略)
```

FFTのライブラリを指定するために追加したオプション

FFTクラスのオブジェクトとしては，二つ宣言しています．一つはcFFT(N)で，これは通常のFFTを行うオブジェクトです．もう一つはcIFFT(-N)で，これは逆FFTを行うオブジェクトです．

データは，複素データとするのでComplex型のデータの実部，および虚部にuRandクラスで生成した乱数を渡します．なお，uRandクラスで生成した乱数は，負の値をとらないので，各データから0.5を引き算し，データが正負の両方に分布するようにしています．

元のデータおよび計算されたデータは，CCSのStdoutウィンドウに出力されます．

FFTの結果とDFTの結果を比較することで，FFTが正しく行われていることが確認できます．また，DFTの定義の項で説明したように，元のデータのDFTを計算し，その逆DFTを計算すると，再び元のデータになります．そこで，このことを利用して，このプログラムでは逆FFTが正しく行われていることを確認しています．

なお，このプログラムをビルドする際のリンカで使用するリンカ・コマンド・ファイルには，FFT用のライブラリを指定するためのオプションも追加する必要があります．使用したリンカ・コマンド・ファイルの一部をリスト7に示します．リスト7で，以下の部分がFFT用のライブラリを指定している箇所です．

```
-l c:¥ti¥myprojects¥lib¥myFFT.lib
```

*　　　　*

次回は，今回作成したFFTの基底クラスを継承する，実数データ用のFFTクラスを作成し，これを使ったフィルタの作りかたについて説明します．

みかみ・なおき　職業能力開発総合大学校 情報工学科

x86CPUだけでもマスタしたい

開発技術者のためのアセンブラ入門

第25回 アセンブラ MASM および gas での SSE/SSE2 命令の使いかた

大貫 広幸

SSE/SSE2命令の使い方

今回はまず，これまで説明したSSE/SSE2の命令を実際にMASM，gasでアセンブルする場合について説明します．

● アセンブラのバージョンと SSE2 命令のサポート

これまでこの連載で使用してきたアセンブラのバージョンは，MASMがVer 6.14，gasがVer 2.10.90でした．しかし，このバージョンのアセンブラはMASMもgasも，MMXとSSEの命令までしかサポートしていないため，SSE2のアセンブルは不可能でした．

SSE2をアセンブルするためには，MASMはVer 7.10以降を使い，gasはVer 2.11.90以降を使用する必要があります．

MASMのVer 7は，Visual Studio.NETに付属しています．詳しいバージョンは，MASM Ver 7.00がVisual Studio.NET 2002，MASM Ver 7.10がVisual Studio.NET 2003です．

MASMでSSE2をアセンブルする場合，まだMASM Ver 7.XXにバグがあるようで一部のSSE2命令は正しくアセンブルすることができません．

Ver 7.10のほうが，Ver 7.00よりバグが少ないのですが，まだ，一部にバクがあるようです．そのため，MASMを使用する場合は，Ver 7.00の使用は避け，Ver 7.10を使用するようにしてください．

またこれは朗報ですが，Visual Studio.NETのドキュメントにも，MASMの言語仕様が掲載されるようになりました．残念ながらCPUの命令は掲載されていませんが，MASMで使われるディレクティブは，このドキュメントを見ることで，その使いかたを知ることができます．

● SSE/SSE2 命令とオペランド

MASMもgasも連載第24回掲載の表1と表2に示したインストラクション名が，そのままニモニックとして使用されています．また，gasはMMX命令と同じように，SSE/SSE2命令のニモニックには型を示す文字は付きません．オペランドの書きかたは，これまでのようにMASMとgasで異なります．

(1) MASMの場合

MASMで，SSE/SSE2命令を使用する場合，ソース・ファイルの頭に，次の2行のディレクティブを記述する必要があります．

```
.686
.XMM
```

最初の「.686」でPentiumPro以降のCPUの使用をアセンブラに指示し，次の「.XMM」でMMX，SSE/SSE2命令の使用をアセンブラに指示しています．浮動小数点命令のオペランドは，転送先をDEST，転送元をSOU，8ビット・イミディエイト値をimm8で表すと，

```
DEST, SOU
DEST, SOU, imm8
```

の2オペランドあるは3オペランドのいずれかになります．ただし，3オペランドで記述されるのはSHUF…命令とCMP…命令のみで，ほかの浮動小数点命令は2オペランドで記述します．

64/128ビット SIMD 整数命令のオペランドは，連載第24回に掲載した表7のようになっています．

制御に関する命令は，MASKMOV…，MOVNT…の命令は，「DEST，SOU」の2オペランド，CLFLUSH，PREFETCH命令がバイト・メモリを指定する1オペランド，LDMXCSR/STMXCSR命令が32ビット・メモリを指定する1オペランドです．FXSAVE/FXRSTOR命令も1オペランドですが，16バイトの倍数に配置された512バイトのメモリを指定します．そして，フェンス命令にはオペランドがありません．

オペランドで使用されるXMMレジスタ名は，CPUのマニュアルで使用されているXMM0～XMM7の名称がそのまま使用されています．実際のMASMでの記述例としてリスト1にSSE/SSE2の浮動小数点命令，リスト2に浮動小数点以外の命令を示します．

(2) gasの場合

gasの場合，アセンブラがMMX，SSE/SSE2命令をサポートしていれば，そのままの状態でMMX，SSE/SSE2命令が使用可能となっています．

SSE/SSE2命令を記述する場合も，オペランドはgasの規則のとおり，インテル表記とは逆の順番でオペランドを記述します．たとえば，浮動小数点命令のオペランドは，転送先をDEST，転送元をSOU，8ビット・イミディエイト値をimm8で

リスト 1　MASMでのSSE/SSE2の浮動小数点命令の記述例（MASM Ver 7.10を使用）

```
                                   .686
                                   .xmm
                                   .model flat

00000000                           .data
00000000 00000000                  m32int1 dd      0
00000004  00000002 [               m64int2 dd      2 dup(0)
           00000000
          ]
                                           align   16
00000010  00000004 [               m128ps  real4   4 dup(0.0)
           00000000
          ]
00000020  00000002 [               m64sp2  real4   2 dup(0.0)
           00000000
          ]
00000028 00000000                  m32ss   real4   0.0

                                           align   16
00000030  00000002 [               m128pd  real8   2 dup(0.0)
           0000000000000000
          ]
00000040                           m64sd   real8   0.0
          0000000000000000

00000000                           .code
                                   ; SSE
00000000  0F 28 E7                         movaps     xmm4,xmm7
00000003  0F 28 05                         movaps     xmm0,m128ps
          00000010 R
0000000A  0F 29 0D                         movaps     m128ps,xmm1
          00000010 R
00000011  0F 10 55 04                      movups     xmm2,[ebp+4]
00000015  0F 11 5D 14                      movups     [ebp+4+16],xmm3

00000019  0F 16 25                         movhps     xmm4,qword ptr m64sp2
          00000020 R
00000020  0F 13 2D                         movlps     qword ptr m128ps+4,xmm5
          00000014 R
00000027  0F 12 CE                         movhlps    xmm1,xmm6
0000002A  0F 50 D7                         movmskps   edx,xmm7

0000002D  F3/ 0F 10 D9                     movss      xmm3,xmm1
00000031  F3/ 0F 10 05                     movss      xmm0,m32ss
          00000028 R
00000039  F3/ 0F 11 0D                     movss      m32ss,xmm1
          00000028 R

00000041  0F 2A DC                         cvtpi2ps   xmm3,mm4
00000044  0F 2A 0D                         cvtpi2ps   xmm1,qword ptr m64int2
          00000004 R
0000004B  0F 2D D6                         cvtps2pi   mm2,xmm6
0000004E  0F 2D 1D                         cvtps2pi   mm3,m64sp2
          00000020 R
00000055  0F 2C D6                         cvttps2pi  mm2,xmm6
00000058  0F 2C 1D                         cvttps2pi  mm3,m64sp2
          00000020 R

0000005F  F3/ 0F 2A D8                     cvtsi2ss   xmm3,eax
00000063  F3/ 0F 2A 0D                     cvtsi2ss   xmm1,m32int1
          00000000 R
0000006B  F3/ 0F 2D CE                     cvtss2si   ecx,xmm6
```

注釈：
- .xmm ← SSE/SSE2命令を使用する場合には必ず記述する
- align 16 ← 128ビット・データは，alignを使用して16バイトの倍数のアドレスになるようにする
- movaps xmm4,xmm7 ← 転送先（DEST）／ 転送元（SOU）
- movups ← メモリ上の128ビット・データが，16バイトの倍数のアドレスにない場合は，MOVU…の命令を使用する

```
0000006F  F3/ 0F 2D 15                     cvtss2si   edx,m32ss
          00000028 R
00000077  F3/ 0F 2C CF                     cvttss2si  ecx,xmm7
0000007B  F3/ 0F 2C 15                     cvttss2si  edx,m32ss
          00000028 R

00000083  0F C6 CD E4                      shufps     xmm1,xmm5,11100100b
00000087  0F C6 15                         shufps     xmm2,m128ps,00011011b
          00000010 R 1B

0000008F  0F 58 D9                         addps      xmm3,xmm1
00000092  0F 58 25                         addps      xmm4,m128ps
          00000010 R
00000099  F3/ 0F 5C EA                     subss      xmm5,xmm2
0000009D  F3/ 0F 5C 35                     subss      xmm6,m32ss
          00000028 R

000000A5  0F C2 C1 00                      cmpps      xmm0,xmm1,0
000000A9  0F C2 C1 00                      cmpeqps    xmm0,xmm1
000000AD  0F C2 05                         cmpps      xmm0,m128ps,1
          00000010 R 01
000000B5  0F C2 05                         cmpltps    xmm0,m128ps
          00000010 R 01
000000BD  0F 2F D3                         comiss     xmm2,xmm3
000000C0  0F 2E 25                         ucomiss    xmm4,m32ss
          00000028 R

000000C7  0F 54 F8                         andps      xmm7,xmm0
000000CA  0F 57 15                         xorps      xmm2,m128ps
          00000010 R
                                   ; SSE2
000000D1  66| 0F 28 E7                     movapd     xmm4,xmm7
000000D5  66| 0F 28 05                     movapd     xmm0,m128pd
          00000030  R 00
          00
000000DD  66| 0F 29 0D                     movapd     m128pd,xmm1
          00000030  R 00
          00
000000E5  66| 0F 10 55 04                  movupd     xmm2,[ebp+4]
000000EA  66| 0F 11 5D 14                  movupd     [ebp+4+16],xmm3

000000EF  66| 0F 16 25                     movhpd     xmm4,qword ptr m64sd
          00000040  R 00
          00
000000F7  66| 0F 13 2D                     movlpd     qword ptr m128pd+8,xmm5
          00000038  R 00
          00
000000FF  66| 0F 50 D7                     movmskpd   edx,xmm7

00000103  F2/ 0F 10 D9                     movsd      xmm3,xmm1
00000107  F2/ 0F 10 05                     movsd      xmm0,m64sd
          00000040 R
0000010F  F2/ 0F 11 0D                     movsd      m64sd,xmm1
          00000040 R

00000117  F3/ 0F E6 DD                     cvtdq2pd   xmm3,xmm5
0000011B  F3/ 0F E6 25                     cvtdq2pd   xmm4,qword ptr m64int2
          00000004 R
00000123  F2/ 0F E6 EF                     cvtpd2dq   xmm5,xmm7
00000127  F2/ 0F E6 2D                     cvtpd2dq   xmm5,m128pd
          00000030 R
0000012F  66| 0F E6 EF                     cvttpd2dq  xmm5,xmm7
00000133  66| 0F E6 2D                     cvttpd2dq  xmm5,m128pd
```

注釈：
- shufps xmm1,xmm5,11100100b ← 転送先（DEST）／ 転送元（SOU）／ imm8
- movapd の 00 ← MASMのバグ．128ビットのメモリ参照時の，この二つの00はリスト上表示されているが，実際のオブジェクトには出力されない
- movsd ← MASMのバグ．MASM Ver 7.00ではこの命令は，エラーになり，アセンブルできない

リスト 1　MASMでのSSE/SSE2の浮動小数点命令の記述例（MASM Ver 7.10を使用）（つづき）

```
         00000030  R 00
         00
0000013B 66| 0F 58 D9            addpd      xmm3,xmm1
0000013F 66| 0F 58 25            addpd      xmm4,m128pd
         00000030  R 00
         00
00000147 F2/ 0F 5C EA            subsd      xmm5,xmm2
0000014B F2/ 0F 5C 35            subsd      xmm6,m64sd
         00000040 R

00000153 66| 0F C2 C1 00         cmppd      xmm0,xmm1,0
                        ;        cmpeqpd    xmm0,xmm1
00000158 66| 0F C2 05            cmppd      xmm0,m128pd,1
         00000030  R 00
         00 01
                        ;        cmpltpd    xmm0,m128pd
00000161 66| 0F 2F D3            comisd     xmm2,xmm3
00000165 66| 0F 2E 25            ucomisd    xmm4,m64sd
         00000040  R 00
```

MASMのバグ.☒ MASM Ver 7.00,☒ Ver 7.10の両方で,☒ この命令はエラー☒ となり,アセンブル できない☒

リスト 2　MASMでのSSE/SSE2の浮動小数点命令以外の記述例（MASM Ver 7.10を使用）

```
                                  .686
                                  .xmm

                                  .model flat

00000000                          .data
                                         align   16
00000000  00000004 [              m128ps  real4   4 dup(?)
           00000000
          ]
00000010  00000002 [              m128pd  real8   2 dup(?)
           0000000000000000
          ]
00000020                          m128dqw dq      ?,?
          0000000000000000
          0000000000000000
00000030                          m64qw   dq      ?
          0000000000000000
00000038 00000000                 m32dw   dd      ?
0000003C 00                       m8b     db      0
                                          align   16
00000040  00000200 [              m512byte        db      512 dup(?)
           00
          ]

00000000                          .code
                                  ; SSE
00000000  0F E4 05                        pmulhuw         mm0,m64qw
          00000030 R
00000007  0F E0 05                        pavgb           mm0,m64qw
          00000030 R
0000000E  0F E3 05                        pavgw           mm0,m64qw
          00000030 R
00000015  0F F6 05                        psadbw          mm0,m64qw
          00000030 R
0000001C  0F C5 DB 02                     pextrw          ebx,mm3,2
00000020  0F C4 F0/ 03                    pinsrw          mm6,eax,3
00000024  0F D7 C9                        pmovmskb        ecx,mm1
00000027  0F 70 0D                        pshufw          mm1,m64qw,00011011b
          00000030 R 1B
                                  ;
0000002F  0F F7 CC                        maskmovq        mm1,mm4
00000032  0F E7 2D                        movntq          m64qw,mm5
          00000030 R
00000039  0F 2B 35                        movntps         m128ps,xmm6
          00000000 R
00000040  0F 18 0D                        prefetcht0      m8b
          0000003C R
00000047  0F 18 15                        prefetcht1      m8b
          0000003C R
0000004E  0F 18 1D                        prefetcht2      m8b
          0000003C R
00000055  0F 18 05                        prefetchnta     m8b
          0000003C R
0000005C  0F AE F8                        sfence
0000005F  0F AE 15                        ldmxcsr         m32dw
          00000038 R
00000066  0F AE 1D                        stmxcsr         m32dw
          00000038 R
0000006D  0F AE 05                        fxsave          m512byte
          00000040 R
00000074  0F AE 0D                        fxrstor         m512byte
          00000040 R
                                  ; SSE2
0000007B  66| 0F 6F 05                    movdqa          xmm0,m128dqw
          00000020 R
00000083  66| 0F 7F 0D                    movdqa          m128dqw,xmm1
          00000020 R
0000008B  F3/ 0F 6F 55                    movdqu          xmm2,[ebp+4]
          04
00000090  F3/ 0F 7F 5D                    movdqu          [ebp+4+8],xmm3
          0C

00000095  66| 0F FC 0D                    paddb           xmm1,m128dqw
          00000020 R
0000009D  66| 0F F4 0D                    pmuludq         xmm1,m128dqw
          00000020  R 00
          00
                                  ;
000000A5  0F AE 3D                        clflush         m8b
          0000003C R
000000AC  66| 0F F7 CC                    maskmovdqu      xmm1,xmm4
000000B0  66| 0F E7 2D                    movntdq         m128dqw,xmm5
          00000020 R
000000B8  66| 0F 2B 35                    movntpd         m128pd,xmm6
          00000010  R 00
          00
000000C0  0F C3 15                        movnti          m32dw,edx
          00000038 R
000000C7  0F AE E8                        lfence
000000CA  0F AE F0/                       mfence
000000CD  F3/ 90                          pause
                                  end
```

SSEで追加された☒ 64ビットSIMD整☒ 数命令の記述例☒（pmulhuw～pshufw）

キャッシュ☒ 制御命令☒（maskmovq～prefetcht0）

プリフェッチ命令☒（prefetcht0～prefetchnta）

命令順序付け命令☒（sfence）

ステート 管理命令☒（ldmxcsr～fxrstor）

SSE2で追加☒ された128☒ ビットSIMD☒ 整数命令の☒ 記述例☒（movdqa～pmuludq）

SSE2の☒ キャッシュ☒ 制御命令☒（clflush～movnti）

SSE2の命令順序付け命令☒（lfence, mfence）

pause命令☒（pause）

リスト 3　gasでのSSE/SSE2の浮動小数点命令の記述例(gas Ver 2.11.90を使用)

```
 1                      .data
 2 0000 00000000        m32int1:     .long       0
 3 0004 00000000        m64int2:     .long       0,0
 3      00000000
 4 000c 00000000                     .align      16
 5 0010 00000000        m128ps:      .float      0.0,0.0,0.0,0.0
 5      00000000
 5      00000000
 5      00000000
 6 0020 00000000        m64sp2:      .float      0.0,0.0
 6      00000000
 7 0028 00000000        m32ss:       .float      0.0
 8
 9 002c 00000000                     .align      16
10 0030 00000000        m128pd:      .double     0.0,0.0
10
10      00000000
10      00000000
11 0040 00000000        m64sd:       .double     0.0
11      00000000
12
13 0048 00000000        .text
13      00000000
14                      # SSE
15 0000 0F28E7                   movaps      %xmm7,%xmm4
16 0003 0F280510                 movaps      m128ps,%xmm0
16      000000
17 000a 0F290D10                 movaps      %xmm1,m128ps
17      000000
18 0011 0F105504                 movups      4(%ebp),%xmm2
19 0015 0F115D14                 movups      %xmm3,16+4(%ebp)
20
21 0019 0F162520                 movhps      m64sp2,%xmm4
21      000000
22 0020 0F132D14                 movlps      %xmm5,m128ps+4
22      000000
23 0027 0F12CE                   movhlps     %xmm6,%xmm1
24 002a 0F50D7                   movmskps    %xmm7,%edx
25
26 002d F30F10D9                 movss       %xmm1,%xmm3
27 0031 F30F1005                 movss       m32ss,%xmm0
27      28000000
28 0039 F30F110D                 movss       %xmm1,m32ss
28      28000000
29
30 0041 0F2ADC                   cvtpi2ps    %mm4,%xmm3
31 0044 0F2A0D04                 cvtpi2ps    m64int2,%xmm1
31      000000
32 004b 0F2DD6                   cvtps2pi    %xmm6,%mm2
33 004e 0F2D1D20                 cvtps2pi    m64sp2,%mm3
33      000000
34 0055 0F2CD6                   cvttps2pi   %xmm6,%mm2
35 0058 0F2C1D20                 cvttps2pi   m64sp2,%mm3
35      000000
36
37 005f F30F2AD8                 cvtsi2ss    %eax,%xmm3
38 0063 F30F2A0D                 cvtsi2ss    m32int1,%xmm1
38      00000000
39 006b F30F2DCE                 cvtss2si    %xmm6,%ecx
40 006f F30F2D15                 cvtss2si    m32ss,%edx
40      28000000
41 0077 F30F2CCF                 cvttss2si   %xmm7,%ecx
42 007b F30F2C15                 cvttss2si   m32ss,%edx
42      28000000
43
44 0083 0FC6CDE4                 shufps      $0b11100100,%xmm5,%xmm1
45 0087 0FC61510                 shufps      $0b00011011,m128ps,%xmm2
45      0000001B
46
47 008f 0F58D9                   addps       %xmm1,%xmm3
48 0092 0F582510                 addps       m128ps,%xmm4
48      000000
49 0099 F30F5CEA                 subss       %xmm2,%xmm5
50 009d F30F5C35                 subss       m32ss,%xmm6
50      28000000
51
52 00a5 0FC2C100                 cmpps       $0,%xmm1,%xmm0
53 00a9 0FC2C100                 cmpeqps     %xmm1,%xmm0
54 00ad 0FC20510                 cmpps       $1,m128ps,%xmm0
54      00000001
55 00b5 0FC20510                 cmpltps     m128ps,%xmm0
55      00000001
56 00bd 0F2FD3                   comiss      %xmm3,%xmm2
57 00c0 0F2E2528                 ucomiss     m32ss,%xmm4
57      000000
58
59 00c7 0F54F8                   andps       %xmm0,%xmm7
60 00ca 0F571510                 xorps       m128ps,%xmm2
60      000000
61                      # SSE2
62 00d1 660F28E7                 movapd      %xmm7,%xmm4
63 00d5 660F2805                 movapd      m128pd,%xmm0
63      30000000
64 00dd 660F290D                 movapd      %xmm1,m128pd
64      30000000
65 00e5 660F1055                 movupd      4(%ebp),%xmm2
65      04
66 00ea 660F115D                 movupd      %xmm3,4+16(%ebp)
66      14
67
68 00ef 660F1625                 movhpd      m64sd,%xmm4
68      40000000
69 00f7 660F132D                 movlpd      %xmm5,m128pd+8
69      38000000
70 00ff 660F50D7                 movmskpd    %xmm7,%edx
71
72 0103 F20F10D9                 movsd       %xmm1,%xmm3
73 0107 F20F1005                 movsd       m64sd,%xmm0
73      40000000
74 010f F20F110D                 movsd       %xmm1,m64sd
74      40000000
75
76 0117 F30FE6DD                 cvtdq2pd    %xmm5,%xmm3
77 011b F30FE625                 cvtdq2pd    m64int2,%xmm4
77      04000000
78 0123 F20FE6EF                 cvtpd2dq    %xmm7,%xmm5
79 0127 F20FE62D                 cvtpd2dq    m128pd,%xmm5
79      30000000
80 012f 660FE6EF                 cvttpd2dq   %xmm7,%xmm5
81 0133 660FE62D                 cvttpd2dq   m128pd,%xmm5
81      30000000
82
83 013b 660F58D9                 addpd       %xmm1,%xmm3
84 013f 660F5825                 addpd       m128pd,%xmm4
84      30000000
85 0147 F20F5CEA                 subsd       %xmm2,%xmm5
86 014b F20F5C35                 subsd       m64sd,%xmm6
86      40000000
87
88 0153 660FC2C1                 cmppd       $0,%xmm1,%xmm0
88      00
89 0158 660FC2C1                 cmpeqpd     %xmm1,%xmm0
89      00
90 015d 660FC205                 cmppd       $1,m128pd,%xmm0
90      30000000
90      01
91 0166 660FC205                 cmpltpd     m128pd,%xmm0
91      30000000
91      01
92 016f 660F2FD3                 comisd      %xmm3,%xmm2
93 0173 660F2E25                 ucomisd     m64sd,%xmm4
93      40000000
94 017b 90                  #
```

注記(行 15 の %xmm7,%xmm4 と行 44 の %xmm5,%xmm1 を指す矢印): 転送元(SOU)☒ / 転送先(DEST)☒

注記(行 15〜16 付近): gasではXMM☒レジスタは，☒%xmm0〜%xmm7で指定☒する☒

注記(行 44 の $0b11100100 を指す): imm8

注記(行 94 の 90 を指す): gas Ver 2.11.90では，リストの最後に，このような☒NOP命令に相当する値が挿入され，セクションのサ☒イズをダブル・ワードの倍数にしている☒

リスト 4　gas での SSE/SSE2 の浮動小数点命令以外の記述例（gas Ver 2.11.90 を使用）

```
 1                      .data
 2                                  .align  16
 3 0000 00000000      m128ps:       .float  0.0,0.0,0.0,0.0
 3      00000000
 3      00000000
 3      00000000
 4 0010 00000000      m128pd:       .double 0.0,0.0
 4      00000000
 4      00000000
 4      00000000
 5 0020 00000000      m128dqw:      .octa   0
 5      00000000
 5      00000000
 5      00000000
 6 0030 00000000      m64qw:        .quad   0
 6      00000000
 7 0038 00000000      m32dw:        .long   0
 8 003c 00            m8b:          .byte   0
 9 003d 000000                      .align  16
10 0040 00000000      m512byte:     .space  512,0
10      00000000
10      00000000
10      00000000
10      00000000
11
12                    .text
13                    # SSE
14 0000 0FE40530              pmulhuw         m64qw,%mm0
14      000000
15 0007 0FE00530              pavgb           m64qw,%mm0
15      000000
16 000e 0FE30530              pavgw           m64qw,%mm0
16      000000
17 0015 0FF60530              psadbw          m64qw,%mm0
17      000000
18 001c 0FC5DB02              pextrw          $2,%mm3,%ebx
19 0020 0FC4F003              pinsrw          $3,%eax,%mm6
20 0024 0FD7C9                pmovmskb        %mm1,%ecx
21 0027 0F700D30              pshufw          $0b00011011,
                                                      m64qw,%mm1
21      0000001B
22                    #
23 002f 0FF7CC                maskmovq        %mm4,%mm1
24 0032 0FE72D30              movntq          %mm5,m64qw
24      000000
25 0039 0F2B3500              movntps         %xmm6,m128ps
25      000000
26 0040 0F180D3C              prefetcht0      m8b
26      000000
27 0047 0F18153C              prefetcht1      m8b
27      000000
28 004e 0F181D3C              prefetcht2      m8b
28      000000
29 0055 0F18053C              prefetchnta     m8b
29      000000
30 005c 0FAEF8                sfence
31 005f 0FAE1538              ldmxcsr         m32dw
31      000000
32 0066 0FAE1D38              stmxcsr         m32dw
32      000000
33 006d 0FAE0540              fxsave          m512byte
33      000000
34 0074 0FAE0D40              fxrstor         m512byte
34      000000
35                    # SSE2
36 007b 660F6F05              movdqa          m128dqw,%xmm0
36      20000000
37 0083 660F7F0D              movdqa          %xmm1,m128dqw
37      20000000
38 008b F30F6F55              movdqu          4(%ebp),%xmm2
38      04
39 0090 F30F7F5D              movdqu          %xmm3,4+8(%ebp)
39      0C
40
41 0095 660FFC0D              paddb           m128dqw,%xmm1
41      20000000
42 009d 660FF40D              pmuludq         m128dqw,%xmm1
42      20000000
43                    #
44 00a5 0FAE3D3C              clflush         m8b
44      000000
45 00ac 660FF7CC              maskmovdqu      %xmm4,%xmm1
46 00b0 660FE72D              movntdq         %xmm5,m128dqw
46      20000000
47 00b8 660F2B35              movntpd         %xmm6,m128pd
47      10000000
48 00c0 0FC31538              movnti          %edx,m32dw
48      000000
49 00c7 0FAEE8                lfence
50 00ca 0FAEF0                mfence
51 00cd F390                  pause
52 00cf 90            #
```

表すと，

　　SOU，DEST

　　imm8，SOU，DEST

と記述することになります．また，オペランドで使用される XMM レジスタ名も，gas の規則のとおり，レジスタ名の頭に % を付けて %xmm0，%xmm1，…，%xmm7 と記述します．

gas での記述例としてリスト 3に SSE/SSE2の浮動小数点命令を，リスト 4に浮動小数点以外の命令を示します．

OS と SIMD 命令の関係

x86系 CPU の SIMD 命令を使用する場合，まず CPU が対応している必要があります．次にアセンブラのような開発環境も対応している必要があります．そしてもう一つ，SIMD 命令を使用する場合，考えておかなければならないことがあります．それは，実際に SIMD 命令を組み込んだアプリケーションを実行する環境です．

通常，Windows や Linux といった OS 上でアプリケーションを実行する場合，OS 自体が SIMD 命令をサポートしている必要があります．具体的には，Windows や Linux のようなマルチタスクの環境でアプリケーション・プログラムを実行する場合，タスク（スレッド）を切り替えるとき，メモリ上のデータの保存もそうですが，CPU や FPU 上のレジスタの値をタスク（スレッド）ごとに退避しておく必要があります．

この CPU や FPU 上のレジスタの退避で，MMX レジスタや XMM レジスタも退避していないと SIMD 命令は使用できません．正確には，一つのタスク（スレッド）のみが SIMD 命令を実行するぶんには SIMD 命令の実行は可能ですが，複数のタスク（スレッド）で SIMD 命令が使われると，ほかのタスク（スレッド）が自分が使用していた MMX レジスタや XMM レジスタを壊してしまい，正しく SIMD 命令の実行ができなくなってしまうからです．

MMX の場合は，そこで使用される MMX レジスタが FPU のレジスタ・スタックと共有されていることから，OS が FPU を

サポートしていれば，MMXも使用可能となります．FPUは，初期のx86系CPUからあるものなので，Windows 95や多少古いLinuxでもMMXは使用できます．

しかし，SSE/SSE2の場合は，XMMレジスタとMXCSRレジスタが新たに追加されたため，OSにもSSE/SSE2のサポートが必要になります．そのためか，CPUにも，OSにSSE/SSE2のサポートがない場合は，SSE/SSE2命令を実行できなくする機構が付けられています．具体的には，CPUの制御レジスタCR4のOSFXSRのビットを1にしないとSSE/SSE2命令は実行できません．CR4のOSFXSRビットは，CPUリセットにより0になるため，そのままの状態ではSSE/SSE2命令は実行できません．また，制御レジスタCR4は，OSのような高い特権をもったプログラムでないとアクセスできないため，アプリケーションが勝手にCR4のOSFXSRビットを1にすることもできないわけです．

つまり，このことから，OSがSSE/SSE2をサポートしない限り，CPUにいくらSSE/SSE2の機能があっても，アプリケーションではSSE/SSE2命令は使用できないというわけです．たとえば，Windows 95は古いOSなのでSSE/SSE2をサポートしていません．そのため，CPUがいくらPentium IIIやPentium4でも，Windows 95ではSSE/SSE2命令は無効命令の例外となり，実行することができません．

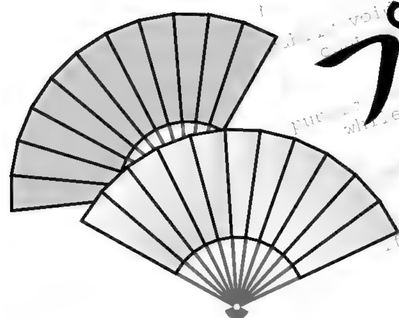
また，SIMD命令から発生する例外に対する例外処理もOSがサポートする必要があります．ただし，MMXは例外を発生させる命令がないので，例外処理も必要ありません．

SSE/SSE2は，連載第23回で述べたようにMXCSRレジスタの例外マスクが0状態で，例外が発生すると割り込みを発生します．そのためSSE/SSE2では，OSによる例外処理のサポートも必要になります．ただし，このOSによる例外処理はオプション的なもので，OSがSSE/SSE2の例外処理を行わない場合は，無効命令の例外として処理することができるようになっています．

具体的には，CPUの制御レジスタCR4のOSXMMEXCPTのビットが0なら，SSE/SSE2の例外割り込みを無効命令の例外（割り込みベクタ番号6）とし，1ならSIMD浮動小数点例外（割り込みベクタ番号19）とします．CPUリセット後のCR4のOSXMMEXCPTビットは0になっているため，OSがSIMD浮動小数点例外をサポートしている場合のみ，OSによりOSXMMEXCPTビットは1に設定されます．

OSがSIMD命令をサポートしているかどうかは，OSの資料をよく調べないとわかりません．おおむねいえることはOSの発売時期とCPUの発売時期で，SIMD命令のサポート・レベルが違ってくるということです．つまり，SIMD命令，とくにSSE/SSE2を使用したければ，CPUのほかにOSも新しいものを使う必要があるということです．

おおぬき・ひろゆき　大貫ソフトウェア設計事務所





プログラミングの要

宮坂 電人

第11回

リスト構造とその応用

双方向連結リスト(doubly-linked lists)

● 単方向連結リストの欠点と双方向連結リスト

通常の配列とは違い，単方向連結リストを使えば，格納するデータの個数がいくら増えても先頭や末尾への追加は瞬時にすむのですが，問題は途中の箇所への追加や削除です．というのも，隣を示すポインタ(あるいは参照)が一つしかないため，そのポインタが示している方向への追加や削除は速いものの，逆方向，すなわちポインタが示していない方向への追加や削除は一筋縄ではいかず，場所を特定するために連結リストの先頭から順番にたどっていく手間が必要になるからです(図1).

しかも，末尾要素を削除しようとすると，先頭からすべての要素をたどっていかざるをえないため，登録しているデータが多くなるほど処理速度は遅くなります．これでは，連結リストを採用した意味がありません．

そこで高速化のために，隣を示すポインタを一つではなく二つ用意したのが双方向連結リストです(図2)．いうまでもなく単方向連結リストと比較すると双方向連結リストはポインター一つ分だけよけいにメモリを消費するので，大量にデータを記録するとメモリ容量の面では不利になります．しかし，連結リストが要求される状況は，たいていの場合，途中箇所への追加や削除が頻繁にあるために配列では不利と判断されたときです．そのため，連結リストを実装するとき，たいていは双方向連結リストとして実装される傾向があります．また，昨今のように安価にメモリを大量に装備できる時代では，ポインター一つ分をケチってパフォーマンスを落とすのは割に合わないと判断されることが多いでしょう．

● 双方向連結リストの実装

単方向連結リストを実装したときと同様，記録したいデータを構造体にし，隣へのポインタ(ここでは二つ)を追加すると汎用性が失われるので，ここでも記録したいデータへの汎用ポインタ(あるいは参照)と隣へのポインタを構造体にします(図3)．Javaで実装する場合，当然のことながら構造体がない

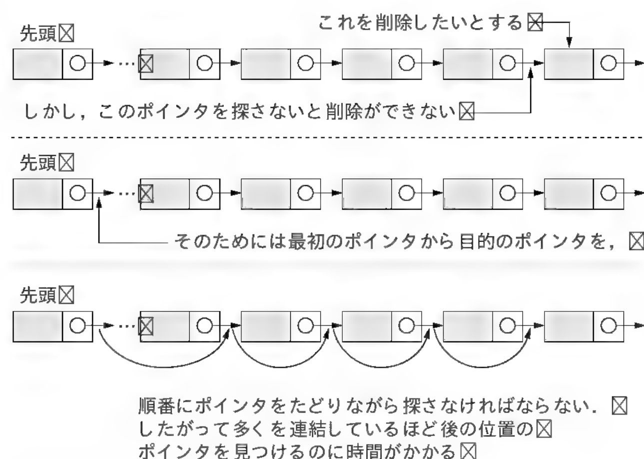


図1　単方向連結リストが不利な例

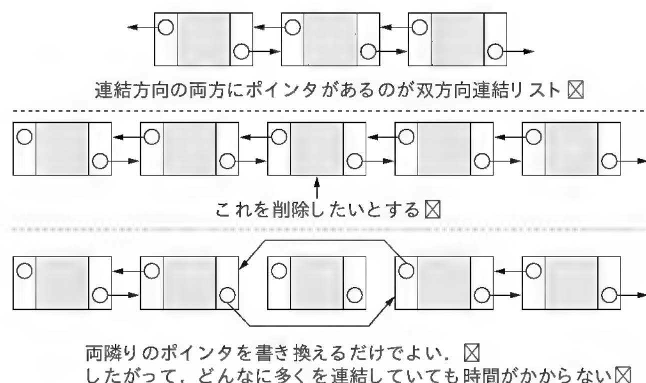


図2　双方向連結リスト

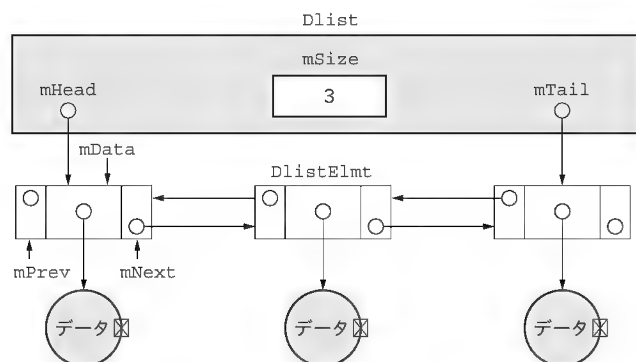


図3　DlistElmtとDlistを使った実装

ので，これをクラスで実現します．記録したいデータへの参照(mData)と隣の要素への参照(mPrev，mNext)は変数にしますが，これへのアクセスは専用メソッドを通して行わせます．具体的にはリスト1のように実装します．

ここで双方向連結リストに必要なサービスを考えてみましょう．リストに記録している要素の個数(mSize)は必須ではありませんが，いくつの要素を記録しているかをすばやく知るために必要になります．これがないとリストの先頭から(あるいは末尾から)要素をたどって個数を勘定しなければならず，当然のことながら処理速度が低下します．また，リストの最初に記録する要素(mHead)は必須です．これがないとそもそも記録や取り出しができなくなります．リストの最後に記録する要素

リスト1　DlistElmt.java(双方向連結リストの実装)

```
public class DlistElmt {

    private Object mData;        // 要素が指すオブジェクト
    private DlistElmt mPrev;     // 前の要素への参照
    private DlistElmt mNext;     // 次の要素への参照

    private DlistElmt() { /* (empty) */ }

    // コンストラクタ iDataは要素が指すオブジェクト
    public DlistElmt(Object iData) {
        mData = iData;
        mPrev = mNext = null;
    }

    // 要素が指すオブジェクトを返す
    public Object getData() {
        return mData;
    }

    // 前の要素への参照をえる
    public DlistElmt getPrev() {
        return mPrev;
    }

    // 次の要素への参照をえる
    public DlistElmt getNext() {
        return mNext;
    }

    // 前の要素への参照をiPrevに変更する
    public void setPrev(DlistElmt iPrev) {
        mPrev = iPrev;
    }

    // 次の要素への参照をiNextに変更する
    public void setNext(DlistElmt iNext) {
        mNext = iNext;
    }
}
```

リスト2　Dlist.java(フィールド)

```
public class Dlist {

    private int mSize;           // 登録している要素の数
    private DlistElmt mHead;     // 先頭の要素への参照
    private DlistElmt mTail;     // 末尾の要素への参照
```

リスト3　Dlist.java(コンストラクタ)

```
    public Dlist() {
        mSize = 0;
        mHead = mTail = null;
    }
```

(mTail)がないと，リストの末尾に記録したいとき，先頭から要素をたどって末尾を調べることになるため処理速度が低下します．以上のより，フィールドはリスト2のようになります．また初期状態はいずれも0あるいはnullになるので，コンストラクタはリスト3のように実装されます．

ここで双方向連結リストで必要になりそうなサービスを考えてみましょう．最初に必要になるのは連結リストに対する登録です．リストの任意の場所に追加するメソッド(insertNext)としてリスト4のように実装します．また双方向連結のおかげで次方向だけではなく前方向に対する挿入が可能です．そこで前方向へ追加するメソッド(insertPrev)としてリスト5を実装します．登録があるなら削除も必要です．リストの任意の場所から削除するメソッド(remove)としてリスト6を実装します．単方向連結リストと違い，削除する対象をポイントする要素を探す必要がなく，削除要素そのものを直接指定できるのが便利なところです．

そのほかのサービスもリスト7のように実装します．

環状連結リスト(circular lists)

● 還状連結リストの使われ方

単方向連結リストと双方向連結リストではリストに登録する要素の先頭と末尾を決めていましたが，これを環状にしてしま

リスト4　Dlist.java(insertNextメソッド)

```
// iElementの次にiDataを追加する
// iElementがnullかつリストが空っぽなら
//   リストの最初の要素にする
public void insertNext(DlistElmt iElement,Object iData) {

    //iElementがnullだがリストが空っぽでないなら処理しない
    if(iElement == null && mSize != 0){
    return;
    }

    //追加したい要素を作成する
    DlistElmt aNewElement = new DlistElmt(iData);

    //追加要素をリストの最初の要素にしたい場合
    if(iElement == null){
        //追加要素は先頭要素かつ末尾要素
        mHead = mTail = aNewElement;
    }else{
        DlistElmt aENext = iElement.getNext();
        //追加要素の次の要素はiElementの次の要素
        aNewElement.setNext(aENext);
        //追加要素の前の要素はiElement
        aNewElement.setPrev(iElement);
        if(aENext == null){ //iElementの次の要素がないなら
            //追加要素は末尾要素でもある
            mTail = aNewElement;
        }else{ //iElementの次の要素があるなら
            //その要素の前の要素は追加要素
            aENext.setPrev(aNewElement);
        }
        //iElementの次の要素は追加要素
        iElement.setNext(aNewElement);
    }

    //登録数を増やす
    ++mSize;
}
```

リスト 5　Dlist.java(insertPrevメソッド)

```
// iElementの前にiDataを追加する
// iElementがnullかつリストが空っぽなら
//  リストの最初の要素にする
public void insertPrev(DlistElmt iElement,Object iData) {

    //iElementがnullだがリストが空っぽでないなら処理しない
    if(iElement == null && mSize != 0){
        return;
    }

    //追加したい要素を作成する
    DlistElmt aNewElement = new DlistElmt(iData);

    //追加要素をリストの最初の要素にしたい場合
    if(iElement == null){
        //追加要素は先頭要素かつ末尾要素
        mHead = mTail = aNewElement;
    }else{ //(この下の部分だけinsertNextと差がある部分)
        DlistElmt aEPrev = iElement.getPrev();
        //追加要素の次の要素はiElement
        aNewElement.setNext(iElement);
        //追加要素の前の要素はiElementの前の要素
        aNewElement.setPrev(aEPrev);
        if(aEPrev == null){ //iElementの前の要素がないなら
            //追加要素は先頭要素でもある
            mHead = aNewElement;
        }else{ //iElementの前の要素があるなら
            //その要素の次の要素は追加要素
            aEPrev.setNext(aNewElement);
        }
        //iElementの前の要素は追加要素
        iElement.setPrev(aNewElement);
    }

    //登録数を増やす
    ++mSize;
}
```

リスト 6　Dlist.java(removeメソッド)

```
// iElementを削除する
// iElementはリストに登録されている要素であること
// 戻り値は削除要素が指していたデータ
public Object remove(DlistElmt iElement) {

    //iElementがnullあるいはリストが空っぽなら処理できない
    if(iElement == null || mSize == 0){
        return null;
    }

    //削除要素の次の要素をえる
    DlistElmt aENext = iElement.getNext();
    //削除しようとする要素が先頭である場合
    if(iElement == mHead){
        mHead = aENext; //先頭要素は削除要素の次の要素
        if(mHead == null){ //先頭要素がなくなった場合
            mTail = null;  //末尾要素は存在しない
        }else{ //先頭要素がなくならない場合
            //先頭より前の要素は存在しない
            mHead.setPrev(null);
        }
    }else{ //削除しようとする要素が先頭でない場合
        //「削除要素の前の要素」の次の要素を
        //「削除要素の次の要素」にする
        iElement.getPrev().setNext(aENext);
        if(aENext == null){ //削除要素の次の要素がない場合
            //末尾要素は削除要素の前の要素
            mTail = iElement.getPrev();
        }else{ //削除要素の次の要素がある場合
            //その要素の前の要素は削除要素の前の要素
            aENext.setPrev(iElement.getPrev());
        }
    }

    //登録数を減らす
    --mSize;
    //削除される要素のデータを返す
    return iElement.getData();
}
```

い，末尾をなくしてしまったのが環状連結リストです．こうすることでリストを順番にたどる作業を無限ループにしてしまう効果があります．

環状連結リストを採用すると便利な例として，通信バッファの実装などが考えられます．通信バッファを配列で固定的に実装するのではなく，環状連結リストで実装すると，バッファ・サイズを後から変えられます．また，通信バッファを単方向連結リストや双方向連結リストで実装すると，メモリの動的確保や解放にともなう実行効率の低下や，確保失敗時の例外処理を心配する必要がありますが，そうしたデメリットから逃れることができます．

● 環状連結リストの実装

「Mastering Algorithms with C」では単方向リストを変形したアプローチで環状連結リストを実装しています(図4)．しかし，必ずしもそのように実装しなければならないわけではありません．個人的には筆者は双方向連結リストを変形したアプローチで実装したほうがよいと考えますが，おそらくは隣の要素を示すポインタが一つ余分に必要なことと，環状連結リストを採用したい局面では途中でリストから要素を削除する機会が少ないであろうことから単方向リストを変形したアプローチにしたのだと想像します．

単方向連結リストを実装したときと同様，記録したいデータへの汎用ポインタ(あるいは参照)と隣へのポインタを一体化し

リスト 7　Dlist.java(そのほかのメソッド)

```
    // 登録数を返す
    public int getSize() {
        return mSize;
    }

    // 先頭要素を返す
    public DlistElmt getHead() {
        return mHead;
    }

    // 末尾要素を返す
    public DlistElmt getTail() {
        return mTail;
    }

    // iElementが先頭要素ならtrueを返す，
    //  そうでないならfalseを返す
    public boolean isHead(DlistElmt iElement) {
        return iElement == mHead;
    }

    // iElementが末尾要素ならtrueを返す，
    //  そうでないならfalseを返す
    public boolean isTail(DlistElmt iElement) {
        return iElement == mTail;
    }
}
```

たクラス(ClistElmt)を用意します．ClistElmtの中身はSlistElmtとまったく同じなので実際のリストは省略します．

ここでも環状連結リストに必要なサービスを考えてみましょう．リストに記録している要素の個数(mSize)とリストのどこ

リスト 8　Clist.java（フィールド）

```
public class Clist {

    private int mSize;        // 登録している要素の数
    private ClistElmt mHead; // 先頭の要素への参照
```

リスト 9　Clist.java（コンストラクタ）

```
public Clist() {
    mSize = 0;
   mHead = null;
}
```

リスト 10　Clist.java（insertNextメソッド）

```
// iElementの次にiDataを追加する
// iElementがnullかつリストが空っぽなら
//  リストの最初の要素にする
public void insertNext(ClistElmt iElement,Object iData) {

    //iElementがnullだがリストが空っぽでないなら処理しない
    if(iElement == null && mSize != 0){
        return;
    }

    //追加したい要素を作成する
    ClistElmt aNewElement = new ClistElmt(iData);

    if(mSize == 0){ //まだ何も登録されていない場合
        //追加要素の次の要素は追加要素となる
        aNewElement.setNext(aNewElement);
        mHead = aNewElement; //先頭要素は追加要素とする
    }else{ //すでに何らかの要素が登録されている場合
        //追加要素の次の要素をiElementの次の要素とする
        aNewElement.setNext(iElement.getNext());
        //iElementの次の要素を追加要素とする
        iElement.setNext(aNewElement);
    }

    //登録数を増やす
    ++mSize;
}
```

リスト 11　Clist.java（removeNextメソッド）

```
// iElementの次の要素を削除する
// iElementはリストに登録されている要素であること
// 戻り値は削除要素が指していたデータ
public Object removeNext(ClistElmt iElement) {

    // 登録数が0あるいはiElementがnullなら削除できない
    if(mSize == 0 || iElement == null){
        return null;
    }

    //削除される予定の要素
    ClistElmt aOldElement = iElement.getNext();

    if(iElement == aOldElement){ //唯一の要素を削除する場合
       mHead = null; //先頭要素はなくなる
    }else{ //要素がまだ残っている場合
        //iElementの次の要素を変更する
        iElement.setNext(aOldElement.getNext());

        //削除される予定の要素が先頭要素なら
        if(aOldElement == mHead){
            //先頭要素を変更しておく
            mHead = aOldElement.getNext();
        }
    }

    //登録数を減らす
    --mSize;
    //削除される要素のデータを返す
    return aOldElement.getData();
}
```

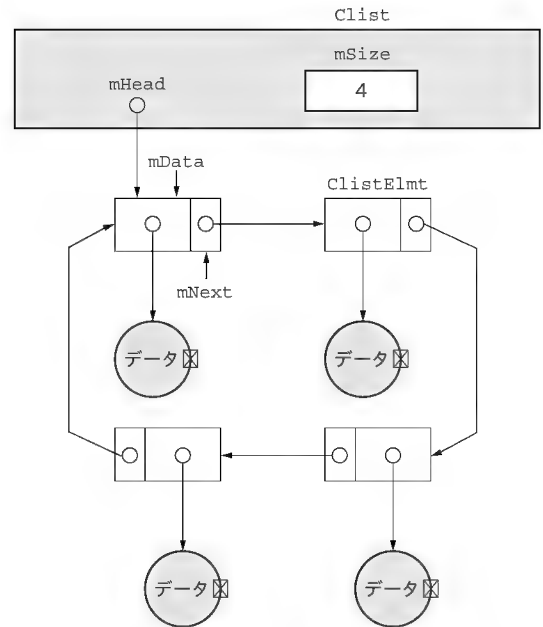


図4　ClistElmtとClistを使った実装

か1点を示す参照（mHead）は必要になるので，フィールドはリスト8のようになります．また，初期状態はいずれも0あるいはnullになるので，コンストラクタはリスト9のように実装されます．単方向連結リストと同様，リストの任意の場所に追加するメソッド（insertNext）としてリスト10を実装します．リストの任意の場所から削除するメソッド（removeNext）としてリスト11を実装します．

スタックとキュー

連結リストができあがれば，基本的なデータ格納機構（コレクション）のうち，スタック（stack，図5）とキュー（queue，図6）が簡単に実装できます．なぜなら，連結リストのしくみ自体がスタックとキューの機能そのものを実現しているからです．

▶スタック：登録したデータがスタックの内部にどんどん追加されていく．データを取り出すときは最後に追加されたものから出てくる．このことから「FILO（First In Last Outの略称．先入れ後出し）」あるいは「LIFO（Last In First Outの略称．後入れ先出し）」とも呼ぶ．

▶キュー：登録したデータがキューの内部にどんどん追加されていく．データを取り出すときは最初に追加されたものから出てくる．このことから「FIFO（First In First Outの略称．先入れ先出し）」とも呼ぶ．

● スタックの実装

わざわざ説明するまでもなく簡単に実装できます．結論からいえば単方向リストを内包したクラスFilo（Javaの標準ライブラリにjava.util.Stackというものがすでに用意されていて，そちらと紛らわしいので名前をFiloに変更した）を用

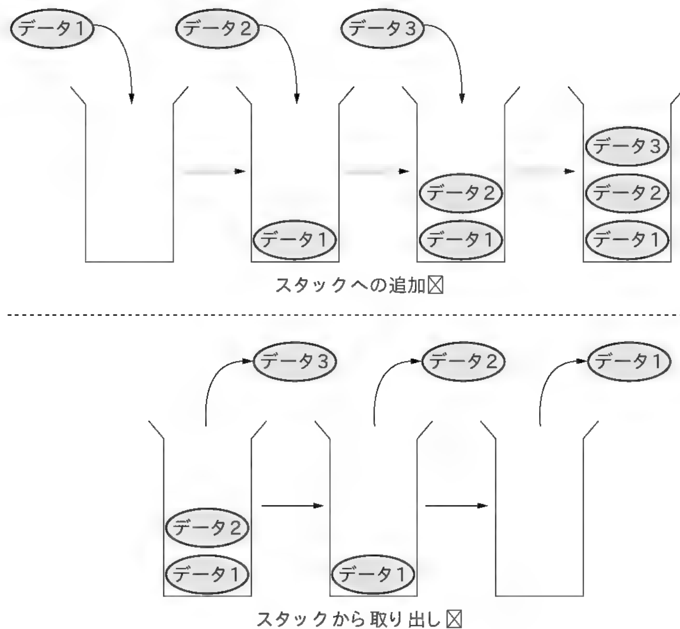


図5　スタックの機能（FILOあるいはLIFO）

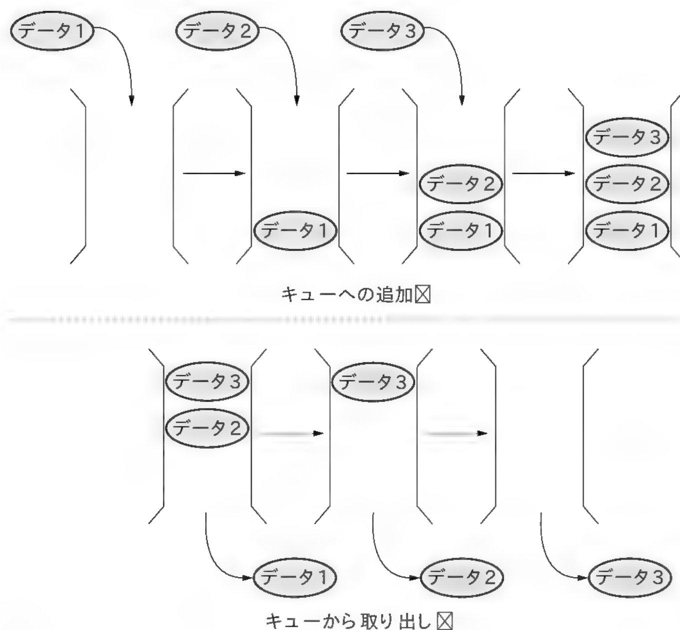


図6　キューの機能（FIFO）

意します．必要なサービスは，

- push：スタックへの追加
- pop　：スタックから取り出し
- peek：スタックの最上部（最後に追加したデータのありか）を見る
- getSize：スタックの登録数をえる

が考えられるので，リスト12のように実装します．

● キューの実装

スタックのときと同様，単方向リストを内包したクラス（Queue）を用意します．必要なサービスは，

- enqueue：キューへの追加
- dequeue：キューから取り出し
- peek：キューの先頭（最初に追加したデータのありか）を見る
- getSize：キューの登録数をえる

が考えられるので，リスト13のように実装します．

みやさか・でんと　miyadent@anet.ne.jp

リスト12　Filo.java（スタックの実装）

```
public class Filo {

    private Slist mSlist;

    // コンストラクタ
    public Filo() {
        mSlist = new Slist();
    }

    //スタックに iData を追加する
    public void push(Object iData) {
        mSlist.insertNext(null,iData);
    }

    //スタックから取り出す
    // 戻り値はスタックの最上部が指していたデータ
    public Object pop(){
        return mSlist.removeNext(null);
    }

    //スタックの最上部が指しているデータを返す
    public Object peek(){
        SlistElmt aTop = mSlist.getHead();
        return (aTop == null) ? null : aTop.getData();
    }

    // 登録数を返す
    public int getSize() {
        return mSlist.getSize();
    }
}
```

リスト13　Queue.java（キューの実装）

```
public class Queue {

    private Slist mSlist;

    // コンストラクタ
    public Queue() {
        mSlist = new Slist();
    }

    // キューに iData を追加する
    public void enqueue(Object iData) {
        mSlist.insertNext(mSlist.getTail(),iData);
    }

    // キューから取り出す
    // 戻り値はキューの先頭が指していたデータ
    public Object dequeue(){
        return mSlist.removeNext(null);
    }

    // キューの先頭が指しているデータを返す
    public Object peek(){
        SlistElmt aTop = mSlist.getHead();
        return (aTop == null) ? null : aTop.getData();
    }

    // 登録数を返す
    public int getSize() {
        return mSlist.getSize();
    }
}
```

Column

巡回メソッドの実装

「Mastering Algorithms with C」ではC言語ベースで実装したため，データ構造の使用終了のときに登録したデータの後始末処理（たとえばメモリの解放など）が必要となります．本連載ではJavaで実装したため，この処理は不要と判断したのですが，ときには後始末処理の内部で行っている登録データの巡回処理が必要となるでしょう．Javaの標準ライブラリでは巡回処理を行いやすいように`java.util.Iterator`が用意されていますが，本連載で作成した`Slist`（単方向連結リスト）と`Dlist`（双方向連結リスト）では要素を表現する`SlistElmt`や`DlistElmt`を使うことで，`java.util.Iterator`の出る幕はありません．たとえば，`Dlist`内部の全登録オブジェクトを表示するにはリストAのようにすればいいわけです．

しかしながら，`java.util.Iterator`ではremoveメソッドのような便利なものもあるうえ，`java.util.Iterator`を使ったプログラムに慣れているなら，あると便利でしょう．試しに`Dlist`に実装したのがリストBです．

先ほどの巡回処理はリストCのようにしても実現できます．

リストA　Dlist内の巡回処理

```
Dlist aDlist;
    ...(略)...
for(DlistElmt aElmt = aDlist.getHead();
     aElmt != null;
     aElmt = aElmt.getNext()){
    Object aObj = aElmt.getData();
    System.out.println("aObj = " + aObj);
}
```

リストC　iteratorメソッドを使った巡回処理

```
Dlist aDlist;
...(略)...
java.util.Iterator aItr = aDlist.iterator();
while(aItr.hasNext()){
    Object aObj = aItr.next();
    System.out.println("aObj = " + aObj);
}
```

リストB　Dlistにiteratorメソッドを追加

```
public class Dlist {
    ...(略)...
    //Dlist専用のイテレータクラス
    private class DlistIterator implements java.util.Iterator {

        private DlistElmt mNextElmt;
        private DlistElmt mPrevElmt;
        private boolean mReverse;
        private Dlist mDlist;

        public DlistIterator(Dlist iDlist,boolean iReverse){
            mDlist = iDlist;
            mReverse = iReverse;
            mNextElmt = mReverse ? iDlist.getTail()
                                  : iDlist.getHead();
            mPrevElmt = null;
        }

        public boolean hasNext(){
            return mNextElmt != null;
        }

        public Object next(){
            if(mNextElmt == null){
                throw new java.util.NoSuchElementException();
            }
            mPrevElmt = mNextElmt;
            mNextElmt = mReverse ? mNextElmt.getPrev()
                                  : mNextElmt.getNext();
            return mPrevElmt.getData();
        }

        public void remove(){
            if(mPrevElmt == null){
                throw new java.lang.IllegalStateException();
            }
            mDlist.remove(mPrevElmt);
            mPrevElmt = null;
        }
    }

    //先頭から末尾に向かうイテレータを取得する
    public java.util.Iterator iterator() {
        return new DlistIterator(this,false);
    }

    //末尾から先頭に向かうイテレータを取得する
    public java.util.Iterator reverse_iterator() {
        return new DlistIterator(this,true);
    }
}
```

組み込みプログラミング・ノウハウ入門(第14回)

アクティブ・オブジェクト・モデリングのこころ

順序集合分割法【その2】

藤倉 俊幸

前回は，ハッセ図を分解せずに一つのアクティブ・オブジェクトとして実装する方法を説明した．今回はハッセ図を分解してアクティブ・オブジェクトを抽出する方法を説明する．

比較のため，前回と同一のハッセ図(図1)を使用して説明する．この図では，各ノードがシステムで発生する事象を表し，矢印がそれらの事象の順序を表す．たとえば，①の事象と②の事象が起きてから④の事象が起きるということを表している．

このような順序を仕様として与えられたときにどのようにしてプログラムを作るのか，ということが前回からの目的である．

前回紹介した方法は，図1のハッセ図を分割せずに，タスクまたはアクティブ・オブジェクトを一つ導入すれば良いので，リソースの少ないシステムの場合には有効である．しかし，一つの処理時間が長いなどの理由からプリエンプションを使用したい場合には，並列度に応じた数のアクティブ・オブジェクトを導入しなければならない．

そのためには，図1のハッセ図を分割する必要が生じる．つまり，分割して，それぞれの事象の処理を別のスレッドに割り当てることで並行実行を可能にし，プリエンプションを利用することでデッド・ラインを守れるようにするわけである．では，どのように分割すれば良いのか，どのように分割すると悲惨な目に遭うのかという点が今回のテーマである．

1 ハッセ図の構造

● 鎖——縦につながった事象の部分集合

順序集合の部分集合で要素すべてに順序関係があるようなものを鎖(chain)と呼ぶ．ハッセ図でいうと縦につながった事象の部分集合が鎖になる．

たとえば，図1のノードの部分集合{2, 4, 6, 8}は鎖である．要素のすべてに順序関係があるという意味は，その部分集合からあらゆる組み合わせで二つを取り出した場合，どの組み合わせにおいてもかならず一方が先でもう一方が後になるということである．

実際に，{2, 4, 6, 8}から2と4を取り出すと2→4，4と8なら4→8といったぐあいである．要するに，ハッセ図上では1本の鎖状になるような部分集合である．

● 反鎖——どちらが先か後かを決められない部分集合

逆に，どのような組み合わせで二つを取り出して比較してもどちらが先か後かが決められないような部分集合を反鎖(antichain)と呼ぶ．

たとえば，{3, 4, 5}や{9, 6, 5}などは反鎖である．順序関係がないので，ハッセ図上では直接線でつながっていない要素だけを集めた部分集合になる．反鎖に含まれる要素は，順序が指定されていないので，どのような順にでも実行できる．つまり並行実行できる．

ちなみに，{2, 3, 4, 5}は鎖でも反鎖でもない，単なる部分集合である．

● ディルワース(Dilworth)の定理[1]

ディルワース(Dilworth)の定理では，

A：「任意の順序集合Pに対して，反鎖の大きさの最大は，互いに素な鎖への分割の最小数に等しい」

ということがわかっている．「大きさ」とは要素の数のことで，「互いに素」とは共通の要素を含まないという意味である．

離散数学の本では，別のいいかたを用いて，

B：「順序集合Pに含まれる最長鎖の長さをnとすると，Pの要素はn個の互いに素な反鎖に分割される」

と紹介されていることもある．

このとき，「長さ」は「大きさ」と同じ意味で使っている．最大反鎖集合の要素数は最小の鎖分割数になり，最大鎖集合の要素の数は最小の反鎖分割数になる．双対関係にあるこれら二つを

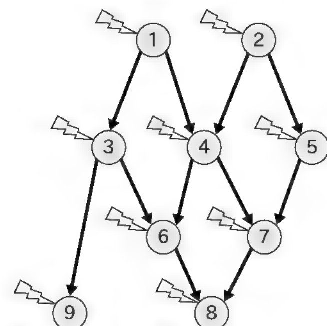


図1 サンプルのハッセ図

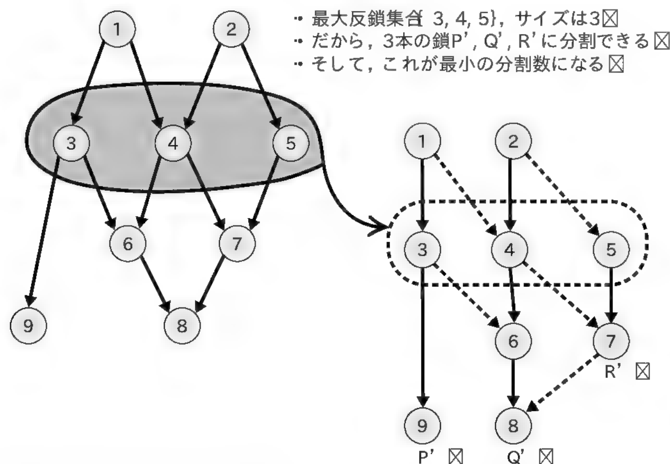


図2　ディルワースの定理 A

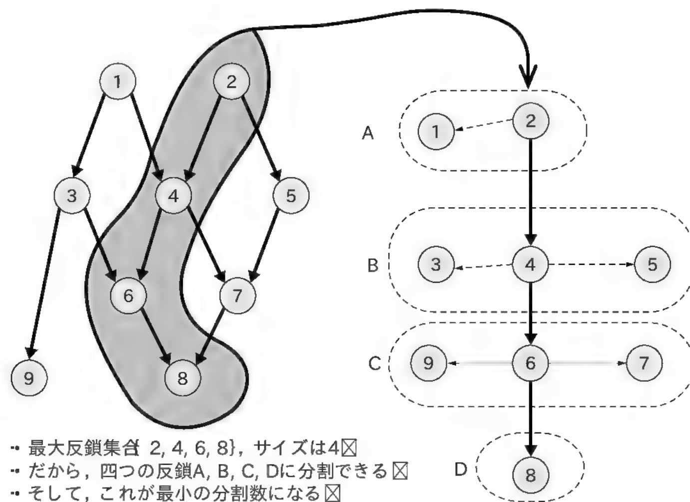


図3　ディルワースの定理 B

図示すると図2と図3のようになる．分割の方法は，一般に何通りもある．

我々が興味をもっているのは，図2のほうである．これが，分析の結果得られたハッセ図の最大並列度と最小タスク数，あるいは最小アクティブ・オブジェクト数の関係になる．

ちなみに，図3は，トポロジカル・ソートなどを考えるときにはつごうが良い．

● 最小鎖分割――よけいな同期構造の導入を極力避ける

並列に実行できる要素の最大数とは，CPUやタスク，アクティブ・オブジェクトなどの並列化要素の数に対応する．

そして，そのときの，それぞれの鎖に含まれている事象が，それぞれの並列化単位への仕事の割り当てになる．

たとえば，図2でP'をタスクとすると，事象1，3，9を処理することが割り当てられる．したがって，図2の場合はP'，Q'，R'の三つの並列化単位を導入すればアプリケーション・ドメインに本質的に含まれている並列性を確保して全事象を処理できることになる．

並列化単位を少なくすると，よけいな順序を導入することになる．

前回は，同一のハッセ図から出発して一つのアクティブ・オブジェクトで実装する方法を説明した．前回の例では，並列性と順序制約を一つのステート・マシンの中に作り込んでしまったわけである．

一つのステート・マシンで実装したのでプリエンプションを実行できないが，処理の単位ではよけいな順序を導入していない．ステート・マシンを使用しない静的クラスのメソッドで実装する場合には，おそらくよけいな順序を設計・実装の段階で付加することになるだろう．

その場合，実世界においたときに外部イベントの順番との間で不整合を起こすようになる．

逆に多くすると，並列化してはいけない処理が並列に走るようになってしまうため，順序制約を守れなくなる可能性がある．そのような過剰な並列性を制御するために，よけいな排他制御を導入しなければならなくなる．そうすると，プログラムは複雑化し，デバッグが泥沼化する場合が多い．

以下の話では，並列化単位をアクティブ・オブジェクトとして扱う．そして，三つのアクティブ・オブジェクトに分ければ，ひとつひとつのアクティブ・オブジェクトを別々のスレッドに割り当てて，プリエンプティブな並行性を導入できるようになるというMDAフレームワークを想定している．

そのようなフレームワークを想定できない場合には，並列化単位[注1]をスレッドと考えればよい．どのようなときに分割の効果があるのかというと，処理時間の長い処理がある際に，ほかの処理の応答性を改善することができる．

処理時間の長い処理を別スレッドに割り当てる手法はよく使われるが，単純にその処理だけを別のスレッドにすると，その処理はいつでも起動可能となり，順序制約を実装するためによけいな同期構造を導入しなければならなくなる．鎖分割(chain partition)することで，よけいな同期構造の導入を極力避けることができる．

ハッセ図の最小鎖への分割は，グラフ理論の最大マッチングを使うことで自動化できる．自動化の例を右掲のコラムにまとめた．

2 分割とアクティブ・オブジェクト

鎖と反鎖，およびステート・マシンの構造の一般的な関係を図4に示す．

反鎖方向に分割して一つのアクティブ・オブジェクトを作った場合，そのアクティブ・オブジェクトのステート・マシンに

注1：したがって，以下では，並列性ということばを厳密には並行性と読み換えてもらいたい．

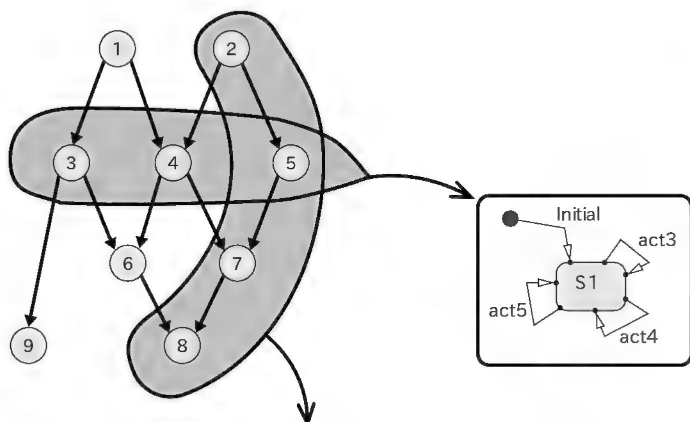


図4　鎖と反鎖，ステート・マシン

は自己遷移が多く含まれるようになる．

一方，鎖方向に分割した場合，ステート・マシンの中にも鎖構造が含まれるようになる．

自己遷移がやたらと多いステート・マシンを作ってしまったような場合には，オブジェクト分割を見直したほうが良い場合がある．動的構造分析を行わないで，デバイスや論理オブジェクトの静的な関係のみでアクティブ・オブジェクトの設計を行った場合，自己遷移だらけのステート・マシンが出現することが多くある．

このようなモデルでは，同期構造のスパゲティ化が始まっている可能性がある．これは変数アクセスや関数呼び出しのスパゲティ化と違って，目に見えないのでやっかいである．

●論理的構造から☒
- ▶一般的なオブジェクト指向のクラス抽出結果を利用する☒
- ☒▶役割ベースでまとめる☒
- ☒・一つの役割を実現するための処理を同一アクティブ・オブジェクトに割り付ける☒

●動的構造から☒
- ▶一般的には鎖に沿って分割☒
- ☒・同期メッセージ数などがオーバ・ヘッドになる☒
- ・鎖に沿って分割することでオーバ・ヘッドを減らせる☒
- ▶最大マッチングを参考にする☒
☒

図5　分割のガイドライン

● 分割

一般的な分割のガイドラインを図5に示す．

従来のオブジェクト指向は，論理的なデータ構造のみから分割するので（アスペクト指向のことばでは，支配的な分割がデータ構造），動的構造が重要な組み込みソフトウェアにオブジェクト指向を適用するのは難しいといわれていた．組み込みシステムの中でも単純な処理しか行なわないものでは，論理的構造を無視して最大マッチングによる自動分割で十分なことも多い．

この場合，オブジェクト指向をまったく使わずに，関数のかたまりとして論理的処理を記述して，それらの関数の呼び出し機構に対してのみアクティブ・オブジェクトを使うスタイルに

Column 1　自動鎖分割＝自動アクティブオブジェクト抽出

鎖分割を自動化する例として，Mathematicaを紹介する．まず，順序対によりグラフ構造rgを作る．

```
rg = FromOrderedPairs[{{3,1},{9,3},{4,1},
    {4,2},{5,2},{6,3},{6,4},{7,4},{7,5},
    {8,6},{8,7}}];
```

次に，そのグラフをハッセ図として表示させる．

```
ShowLabeledGraph[ HasseDiagram[rg] ];
```

これで本文と同一のハッセ図を作ることができる（図A）．このハッセ図について，

```
MinimumChainPartition[rg]
```

を実行すると，

```
{{1,3,9},{2,4,6,8},{5,7}}
```

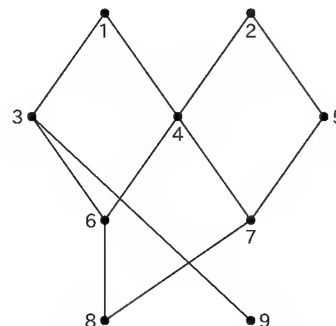
と出力され，最大マッチングを利用して鎖分割を得ることができる．これが，自動化されたアクティブ・オブジェクト分割に対応する．

最大反鎖を求める場合には，次のようにする．

```
MaximumAntichain[rg]

{3,4,5}
```

実際の設計では，最大マッチングで鎖分割まで作るよりは最大反鎖をもとにして，自分で試行錯誤しながら鎖を伸ばすことで分割を作ることが多い．



図A
Mathematicaで作成したハッセ図

なる．支配的な分割を動的構造にしてしまうスタイルである．

論理的構造を無視できない場合には，アスペクトを利用するかレイヤ構造，あるいは本連載の第3回で取り上げたクライアント-サーバ・アーキテクチャを組み合わせて利用する．

ちなみに，ここで説明している鎖分割によって作られるソフトウェア・アーキテクチャは，第3回で取り上げたパイプライン型アーキテクチャに対応する．

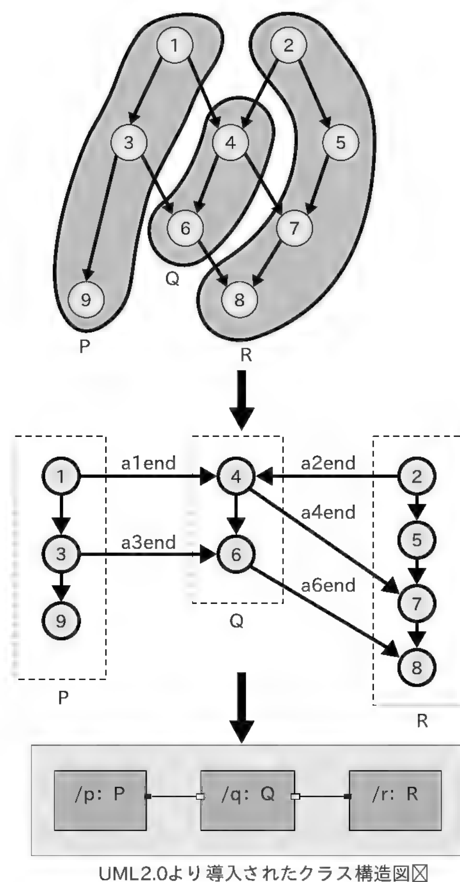


図6　同期用インターフェース定義

● 同期インターフェース

ハッセ図の分割が決まったら，アクティブ・オブジェクト間の同期インターフェースを決める．分割内の順序制約はステート・マシンの中に作り込まれるので，同期インターフェースが必要になるのは分割の境界線上の順序制約である．

図6に示すP，Q，Rに分割する場合では，a1endとa2end，a3end，a4end，a6endを同期インターフェースとして定義する必要がある．

次に，これらの内部メッセージの通信路を構造図の形で表現する．この構造図は，UML2.0で採用される予定のアーキテクチャを表現するための新しいダイヤグラムである．構造図によってアクティブ・オブジェクトと通信路を表現することができる．

図2に示した分割方法の場合でも追加する同期用内部メッセージの数は変わらないが，種類が異なり，a1endとa2end，a3end，a4end，a7endになる．

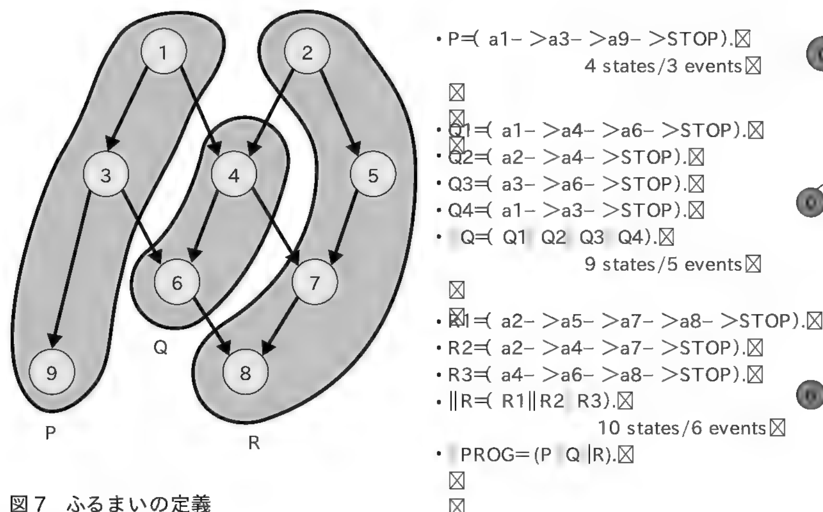
最大マッチングによる最小分割を行っていれば，追加しなければならない内部メッセージの数は同一である．図6と図2のどちらが良いのかという点については，たいていは静的なオブジェクト指向分析から得られる各処理の論理的な結合度によって決める．

しかし，論理的結合度を無視して，ふるまいを含んだ動的な複雑度を下げたほうが良い場合もある(レイヤ構造を基本としてアスペクトをところどころに導入するアプローチ)．

● アーキテクチャに対応するふるまい

アーキテクチャ(分割とインターフェース)が決まったら，次は個々のアクティブ・オブジェクトのふるまいをステート・マシンで定義する．図6のアーキテクチャに対応するふるまいの定義を図7に示す．プロセス代数を使用したステート・マシンの生成方法は前回説明したとおりである．

この例の場合，外部イベントはa1からa9の9種類ある．重複を考えないですべてのイベント順のブラック・ボックス・テ



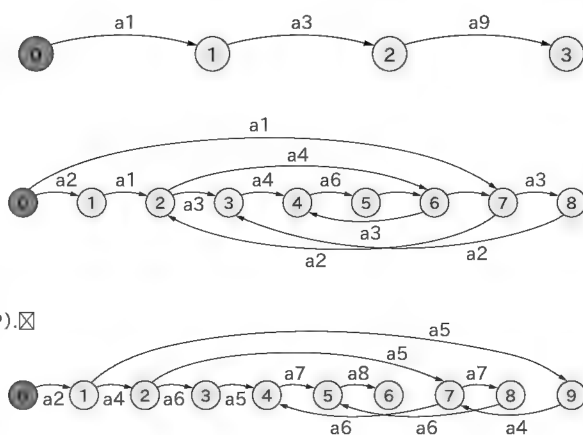


図7　ふるまいの定義

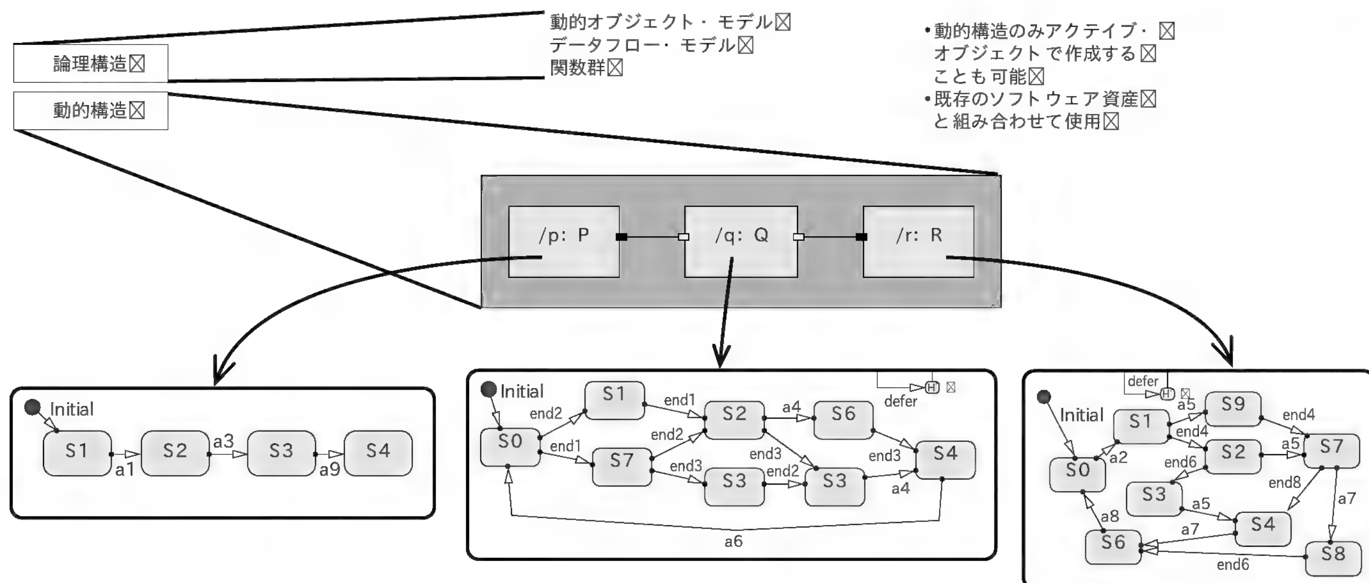


図8 動的構造の利用

ストを実施しようとすると，

9！＝362880通り

という数の試験が必要になる．前回の一つのアクティブ・オブジェクトで実装した場合，オブジェクト単体テストとして，これだけのイベント順のテストが必要になる．

しかし，三つに分割したことで必要な単体テストの総数を減らすことができる．図6に示した分割の場合，それぞれのアクティブ・オブジェクトP，Q，Rが考慮しなければならないイベントは，

P：a1，a3，a9の3種類

Q：a4，a6，a1end，a2end，a3endの5種類

R：a2，a5，a7，a8，a4end，a6endの6種類

になるので，必要な単体テストの総数は，

3！＋5！＋6！＝846通り

と少なくなり，十分に実施可能な範囲になる．

結合テストは，導入した内部イベントと外部イベントの部分を調べる必要があるので，以下のイベントの組み合わせテストが必要になる．

● 内部イベントの個別テスト

{a1, a2, a4} 3！＝6通り

{a3, a4, a5} 3！＝6通り

{a4, a5, a7} 3！＝6通り

{a6, a7, a8} 3！＝6通り

● 内部イベントの組み合わせテスト

5！＝120通り

全体で144通りなので，単体テストも含めても846＋144＝990通りにすぎない．

これは，分割することで全体の複雑度を下げることができるという例である．

一方，図2の分割の場合の単体テストは，

P'：a1，a3，a9の3種類

Q'：a2，a4，a6，a8，a1end，a3end，a7endの7種類

R'：a5，a7，a2end，a4endの4種類

となり，したがって，

3！＋7！＋4！＝5070通り

と，図6よりも増加してしまう．

また，Q'のステート・マシンも状態数が23状態となり，Q'の複雑度があまり減少していない．そのため，状態とイベントの組み合わせテストまで考えると膨大なテストが必要になる．

したがって，導入しなければならない内部イベントの数などのアーキテクチャ・レベルでは，図6と図2では差がなかったが，ふるまいまで考慮するとかなり差があることがわかる．

ここでは，外部イベントの組み合わせテストを考えたが，ステート・マシンのホワイト・ボックス・テストでは，(状態数−1)だけのイベント列を考慮する必要がある(2)．

したがって，Q'の場合のホワイト・ボックス・テストは，7種類のイベントの重複を許した22イベントの重複順列がテストの総数ということになる．つまり，状態数の偏りがないように等しく分散されるようなアクティブ・オブジェクト分割が，ホワイト・ボックス的には有利である．

静的なオブジェクト指向分析設計だけでアクティブ・オブジェクトを利用すると，このあたりの差は詳細設計に入るまで見えてこない．

役割ベースの論理的分割と動的分割が一致しない場合には，ここで設計したアクティブ・オブジェクトによる動的構造は，図8に示すように一つのレイヤとして実装する．そして，論理

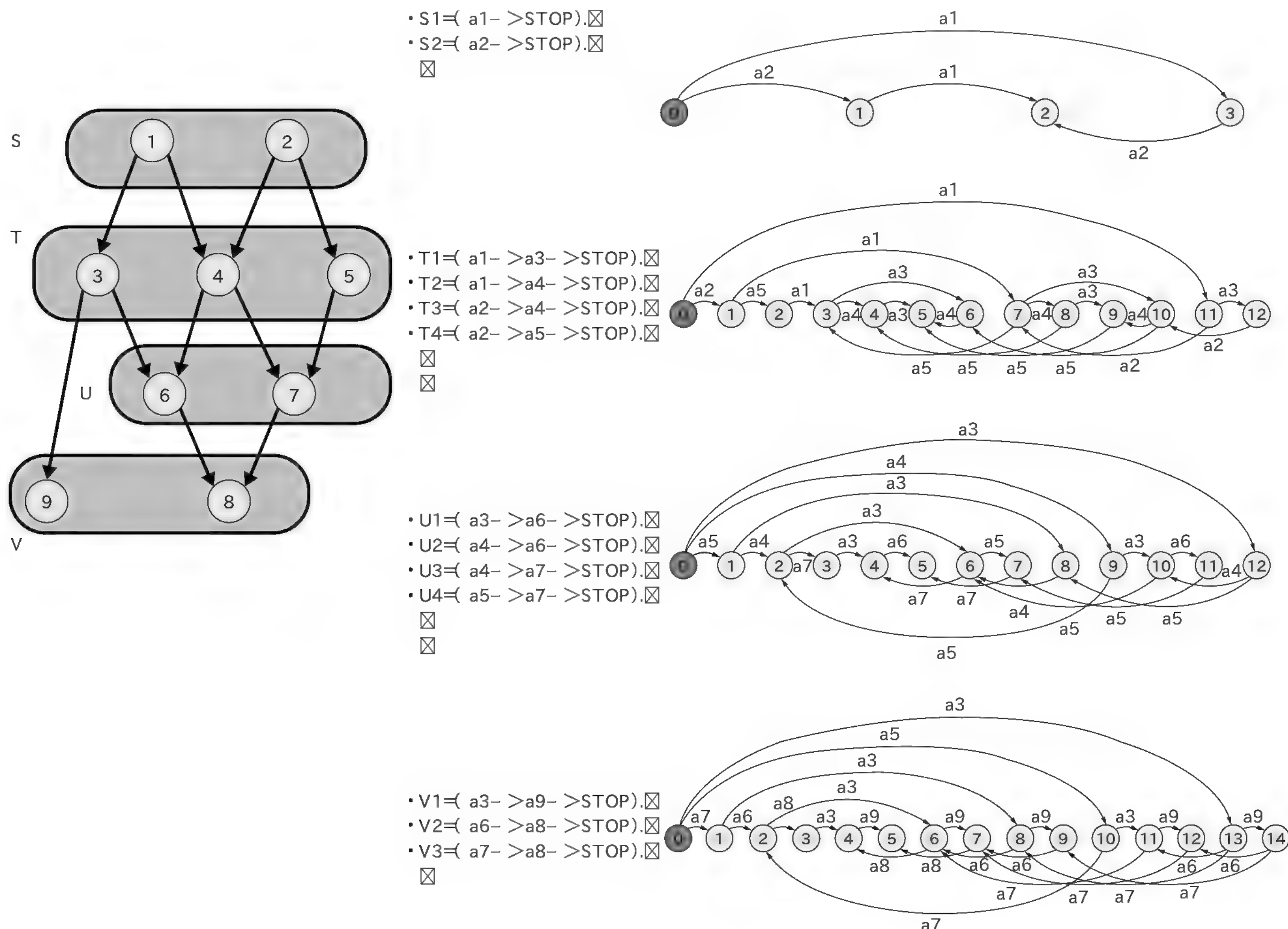


図9　反鎖による分割の例

的構造を優先させたい場合には，一致しない部分はアスペクトやコールバックなどを利用して実装する．

3 反鎖による分割

図9に動的構造を無視して反鎖による分割を行った場合の各アクティブ・オブジェクトのふるまいを示す．

このような分割では，スレッドを導入してもアプリケーションに本質的な並列性を満足するためにプリエンプションを使うことができないということはすでに指摘した．そのほかに，

1) 順序制約すべてを内部同期イベントで実現しているため，イベント・オーバ・ヘッドが増える
2) 各アクティブ・オブジェクトのステート・マシンの複雑度が下がらない

などといった問題もある．そして，内部イベントの組み合わせテストだけで，

11！ = 39916800通り

という数になってしまうので，元(もと)の木阿弥(もくあみ)どころか逆に複雑さを作り込む結果になっている．

動的構造を把握しないと，いつの間にか反鎖による分割を行っている場合が多くなるので注意しなければならない．

4 そのままモデリング

オブジェクト指向は，本質的にはデータの囲い込みなので，同じデータにアクセスする関数が一つのクラスにまとめられる．それから，一般的にオブジェクトを抽出する際には，ドメイン・オブジェクトが第1候補になる．組み込みソフトウェアの場合，ハードウェア構成図などからドメイン・オブジェクトが抽出されるので，図10に示すように，オブジェクト分割が行われる．

この分割は反鎖分割に対応する．反鎖分割は動的構造を複雑化するだけで何のメリットもないということはすでに説明した．そこで，図11に示したように各ドメイン・オブジェクトの中の事象に注目し，事象間の先行制約を分析しなければならない．

組み込み用のオブジェクト指向を標榜する著作や方法論がい

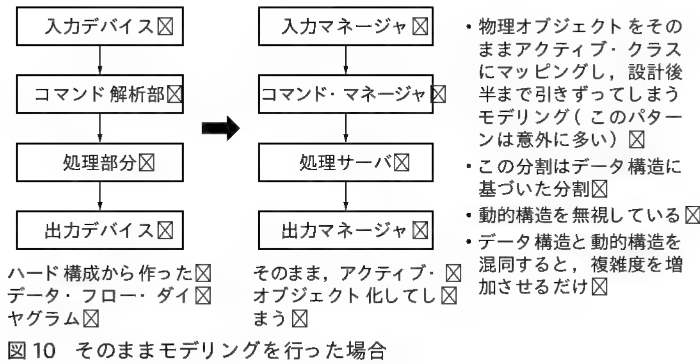
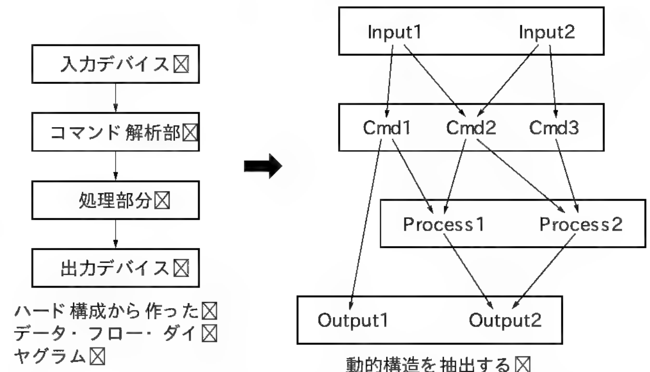


図10　そのままモデリングを行った場合



- 安易に物理オブジェクトをそのままアクティブ・クラスにマッピングしない
- オブジェクトの中の事象の順序に注目する
- そうすれば動的構造が見えてくる
- その動的構造を最小鎖分割してアクティブ・オブジェクトとする

図11　動的構造分析

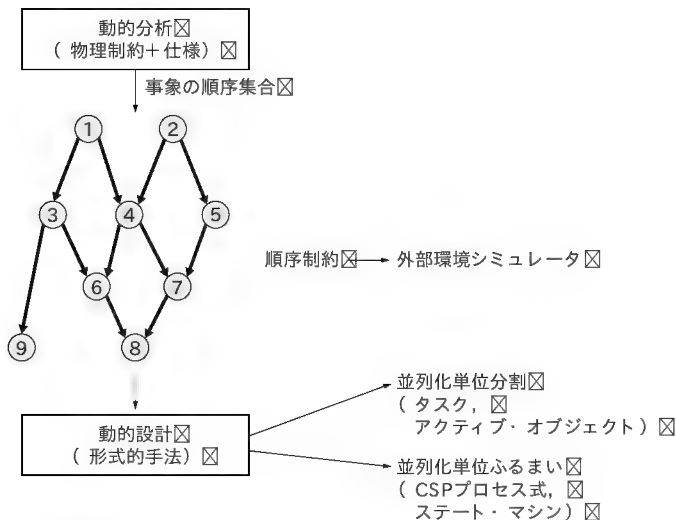


図12　情報の流れ

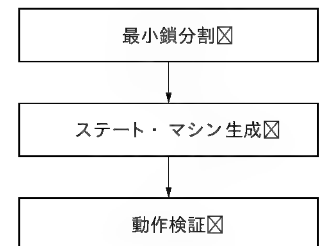


図13　動的設計手順

くつかあるが，図10のレベルのものがほとんどである．役割による分割のみでは複雑な動的構造を持った組み込みシステムではほとんどの場合，失敗すると思われる．

まとめ

今回と前回の2回にわたって，手法の概要を説明した．情報の流れを図12に示す．

次回は，実際の問題に適用する例をいくつか示す．また，図13の動作検証に対応するsafetyとlivenessに関する検証についても今後説明する予定である．

図10のレベルからもう一歩踏み出して順序制約を抽出するだけで，後はグラフ理論やプロセス代数などの形式的手法により，検証済みの実装まで進むことができる．これが，アクティブ・オブジェクト・モデリングの心である．

参考文献

(1) R.P.Dilworth, "*A decomposition theorem for partially ordered set.*", Ann. of Math., 51, pp.160-166, 1950.
(2) T.S.Chow, "*Testing software design modeled by finite-state machines*", IEEE Trans. Software Eng., SE-4, 2, pp.178-187, 1978.

ふじくら・としゆき　組み込みコンサルタント
tfujikura@jcom.home.ne.jp

やり直しのための信号数学 第23回



DCTによる信号処理応用(その2)

三谷 政昭

前回(2004年4月号)は，DCT(ディジタル・コサイン変換)による信号処理応用として1次元データ(主として，音声)を題材に，雑音除去，グラフィック・イコライザ，プッシュ・ホンの電話番号(トーン信号)の送出，選択，認識などについての考えかたを中心に解説した．

今回は，DCTによる信号処理応用の第2弾として2次元データ(主として，画像)処理をとりあげる．まず，1次元DCTを2次元DCTに拡張したあと，具体的な画像処理の適用事例として，DCTを用いた正規直交基底画像による展開や，2次元フィルタによる信号処理(たとえば，雑音除去，輪郭の抽出)などについてわかりやすく説明する．

DCTによる画像データ処理

一般に，DCTはディジタル画像処理の分野で広く利用されている．代表的な適用例としては，画像圧縮，画質改善および画像復元などがある(図23.1)．

● 画像圧縮

画像のひずみを極力抑えながら，同時にできるだけ少ないデータ量で画像を蓄積保存，あるいは伝送しようとするものである．つまり，画像のような2次元信号のデータは音声などの1次元信号に比べると，通常は膨大な量になるので，データ量を削減する必要性に起因している．このような画像の取り扱いはデータ圧縮とも呼ばれ，たとえば，JPEGやMPEGなどに代表される圧縮データ形式によるWeb上の画像表示，ディジタル・テレビ放送など，多種多様な画像処理分野で広汎に利用されている．

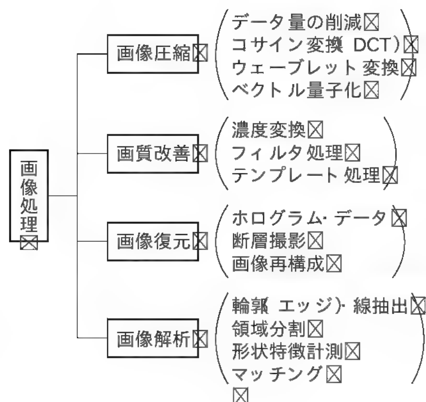


図23.1
画像処理の分野で行われるデータ処理の内容

● 画質改善

劣化している画像に対し，その劣化要因を取り除き，見やすい画像を得るために行われる．たとえば，暗すぎる画像に対して明るくしたいとか，コントラストの悪い画像をシャープなものに改善したいような場合，簡単な濃度変換により実現される．また雑音が多い画像に対しては，平滑化(平均化)などの2次元フィルタ処理を施せばよい．

● 画像復元

対象物の"真"の画像を再現しようとするために行われる．たとえば3次元物体(立体)を多方向から投影したデータから再構成して3D(立体)表示させる．身近なところでは，病院でのX線や超音波による複数枚の断層撮影画像からコンピュータ処理による肺や胃などの内臓の3D表示がそうである．また対象物から画像を生成する際に生ずる原画像の劣化に対処するものとして，カメラ撮影時の手ぶれによる動きの補償，レンズの焦点ずれや収差のためにぼけたり歪んだ像の補正なども画像復元処理の例である．

画像と2次元データによる表現

まず，写真やビデオ映像からディジタル画像を得る処理プロセスを図23.2に示す．たとえば，写真の場合は直線上にCCDなどの受光素子(ライン・センサ)を配置したイメージ・スキャナが用いられ，横方向は電子的に，縦方向は機械的に走査され



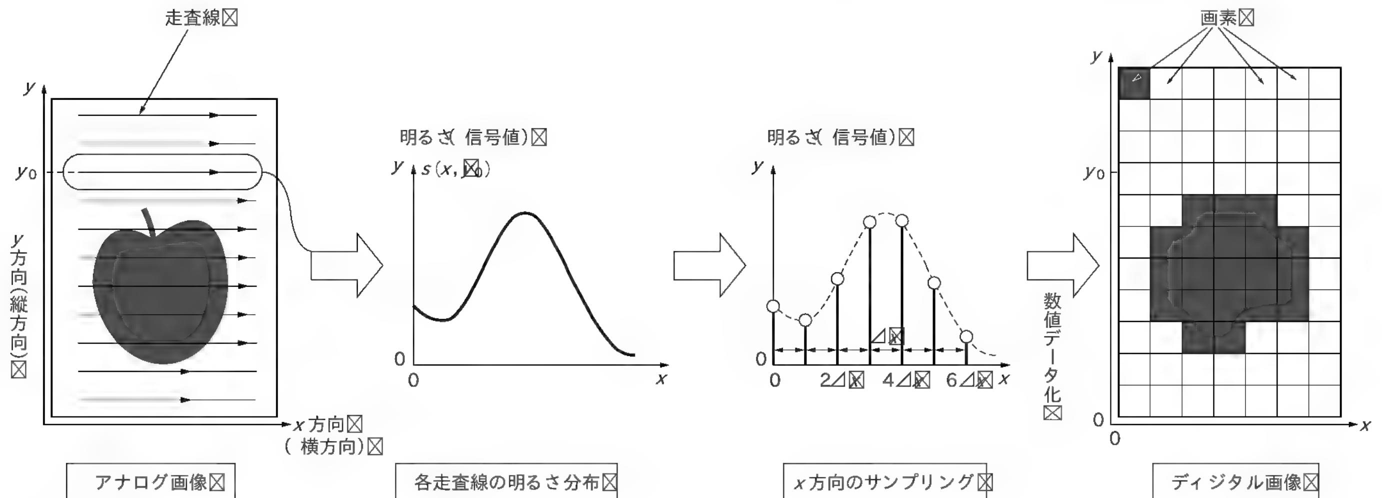


図 23.2　ディジタル画像を得る処理の流れ

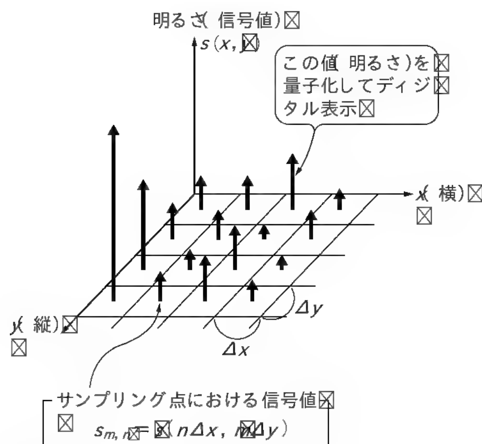


図 23.3　画像データのディジタル化

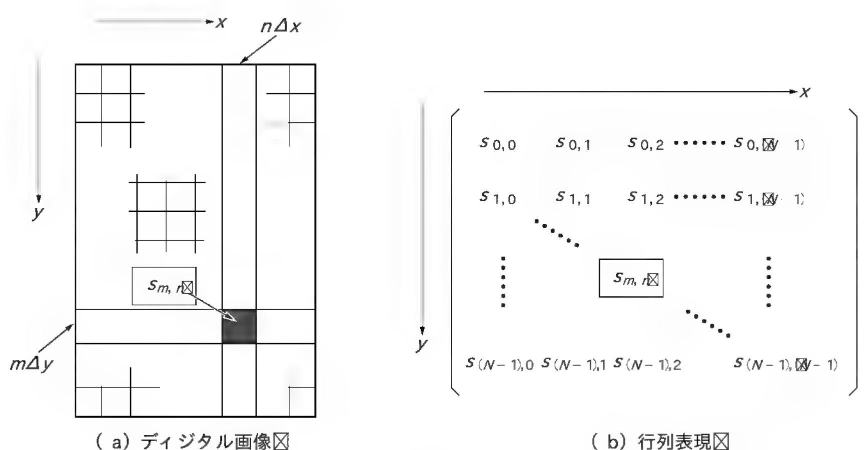


図 23.4　ディジタル画像と行列（配列）表現の対比

る．また，ディジタル・カメラでは2次元的に撮影された出力信号，すなわちビデオ映像そのものがディジタル信号となる．いずれにしても，得られた画像信号データは2次元的に配置される．

いま，果物のモノクロの静止画像を考えることにする．図23.3のように明るさ（輝度値，濃度値ともいう）が2次元的に変化しており，表示位置を示すx（横方向）とy（縦方向）に関する2次元信号$s(x, y)$として表現される．この画像信号$s(x, y)$をディジタル・データに変換するためには，縦横にそれぞれ間隔Δx，Δyでサンプリングする必要があり，整数m，nを用いて，

$$s_{m,n} = s(n\Delta x, m\Delta y) \quad \cdots\cdots (1)$$

となる．これは，図23.2に示した処理プロセスに基づき，画像を小領域（画素，あるいはピクセルという）に分割して，個々の小領域ごとに明るさを指定したものと考えられる．

たとえば画像サイズが$N \times N$画素のディジタル画像の場合は図23.4のように，画面の左上を$s_{0,0}$，右下を$s_{(N-1)(N-1)}$とするのが一般的である．

■ 2次元DCTの定義と計算手順

いま，図23.4の$N \times N$画素の画像信号$\{s_{m,n}\}_{m,n=0}^{m,n=N-1}$に対する2次元DCT値を，

$$\begin{bmatrix} C_{0,0} & C_{0,1} & C_{0,2} & \cdots & C_{0,N-1} \\ C_{1,0} & C_{1,1} & C_{1,2} & \cdots & C_{1,N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{(N-1),0} & C_{(N-1),1} & C_{(N-1),2} & \cdots & C_{(N-1),(N-1)} \end{bmatrix}$$

と行列表示すると，各要素$\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$は次式で与えられる．

$$C_{k,\ell} = \frac{1}{N^2} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \gamma_k \gamma_\ell s_{m,n} \cos\left[\frac{(2m+1)k}{2N}\pi\right] \cos\left[\frac{(2n+1)\ell}{2N}\pi\right] \quad \cdots\cdots (2)$$

ただし，
$$\gamma_i = \begin{cases} 1; & i = 0 \\ \sqrt{2}; & i \neq 0 \end{cases} \quad \cdots\cdots (3)$$

ところで，筆算による画像データの DCT，IDCT 処理は 2次元データ処理を感覚的につかむことができる．そこで，筆算を簡略化して実行するために，式(2)の 2次元 DCT 値が 1次元 DCT の計算を 2回実行すれば算出できることを示しておきたい．

式(2)の総和計算する処理(Σ)について，

$$C_{k,\ell}=\frac{1}{N}\sum_{n=0}^{N-1}\gamma_\ell\cos\left[\frac{(2n+1)\ell}{2N}\pi\right]\left\{\frac{1}{N}\sum_{m=0}^{N-1}\gamma_k s_{m,n}\cos\left[\frac{(2m+1)k}{2N}\pi\right]\right\} \quad\cdots\cdots(4)$$

と変形すれば，

$$\begin{cases} P_{k,n}=\dfrac{1}{N}\displaystyle\sum_{m=0}^{N-1}\gamma_k s_{m,n}\cos\left[\dfrac{(2m+1)k}{2N}\pi\right] & \cdots\cdots(5)\\ C_{k,\ell}=\dfrac{1}{N}\displaystyle\sum_{n=0}^{N-1}\gamma_\ell P_{k,n}\cos\left[\dfrac{(2n+1)\ell}{2N}\pi\right] & \cdots\cdots(6)\end{cases}$$

と表される．したがって，式(5)の列方向の 1次元 DCT により計算される値 $\{P_{k,n}\}_{k,n=0}^{k,n=N-1}$ に対して，さらに式(6)の行方向の 1次元 DCT を適用すれば，2次元 DCT 計算が実行されたことに等価であることが理解される〔図 23.5(a)〕．

同様に，式(2)の総和計算する処理(Σ)について，

$$C_{k,\ell}=\frac{1}{N}\sum_{m=0}^{N-1}\gamma_k\cos\left[\frac{(2m+1)k}{2N}\pi\right]\left\{\frac{1}{N}\sum_{n=0}^{N-1}\gamma_\ell s_{m,n}\cos\left[\frac{(2n+1)\ell}{2N}\pi\right]\right\} \quad\cdots\cdots(7)$$

と変形すれば，

$$\begin{cases} Q_{m,\ell}=\dfrac{1}{N}\displaystyle\sum_{n=0}^{N-1}\gamma_\ell s_{m,n}\cos\left[\dfrac{(2n+1)\ell}{2N}\pi\right] & \cdots\cdots(8)\\ C_{k,\ell}=\dfrac{1}{N}\displaystyle\sum_{m=0}^{N-1}\gamma_k Q_{m,\ell}\cos\left[\dfrac{(2m+1)k}{2N}\pi\right] & \cdots\cdots(9)\end{cases}$$

と表される．よって，式(8)の行方向の 1次元 DCT により計算される値 $\{Q_{m,\ell}\}_{m,\ell=0}^{m,\ell=N-1}$ に対して，さらに式(9)の列方向の 1次元 DCT を適用すれば，2次元 DCT 計算が実行されるのである〔図 23.5(b)〕．

いま，2×2画素の簡単な画像データ $\{s_{m,n}\}_{m,n=0}^{m,n=1}$ を，

$$\begin{bmatrix} s_{0,0} & s_{0,1}\\ s_{1,0} & s_{1,1}\end{bmatrix}=\begin{bmatrix}14 & 2\\ 4 & 0\end{bmatrix} \quad\cdots\cdots(10)$$

として，2次元 DCT 値，すなわち，

$$\begin{bmatrix} C_{0,0} & C_{0,1}\\ C_{1,0} & C_{1,1}\end{bmatrix} \quad\cdots\cdots(11)$$

を求めてみよう．

● 2 次元 DCT 計算式

式(2)～式(9)において，N=2を代入することにより得られる 2次元 DCT の 3通りの計算式を以下に示す．

○ 式(2)による直接計算

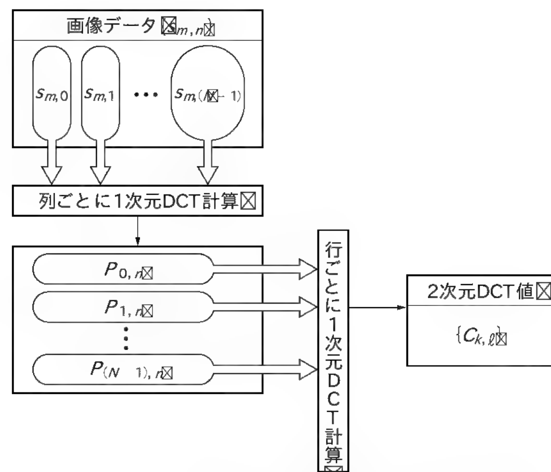
$$\begin{cases} C_{0,0}=\dfrac{s_{0,0}+s_{0,1}+s_{1,0}+s_{1,1}}{4}\\ C_{0,1}=\dfrac{s_{0,0}-s_{0,1}+s_{1,0}-s_{1,1}}{4}\\ C_{1,0}=\dfrac{s_{0,0}+s_{0,1}-s_{1,0}-s_{1,1}}{4}\\ C_{1,1}=\dfrac{s_{0,0}-s_{0,1}-s_{1,0}+s_{1,1}}{4}\end{cases} \quad\cdots\cdots(12)$$

○ 式(4)～式(6)による 2ステップ計算

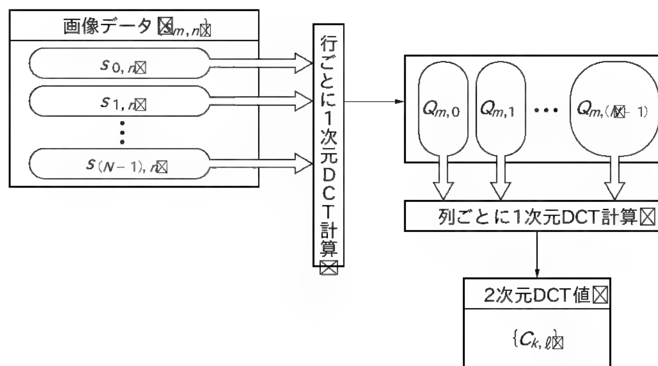
まず，列(縦)方向の二つの画素を 1ブロックとし，1次元 DCT 値を求める．この得られた 1次元 DCT 値の行(横)方向の二つをまとめて 1ブロックとし，1次元 DCT 値を求める．

● 列ブロックごとの計算

$$\begin{cases} P_{0,0}=\dfrac{s_{0,0}+s_{1,0}}{2},\ P_{0,1}=\dfrac{s_{0,1}+s_{1,1}}{2}\\ P_{1,0}=\dfrac{s_{0,0}-s_{1,0}}{2},\ P_{1,1}=\dfrac{s_{0,1}-s_{1,1}}{2}\end{cases} \quad\cdots\cdots(13)$$



(a) 列→行の順による2次元DCT計算



(b) 行→列の順による2次元DCT計算

図 23.5　1次元 DCT を適用した 2次元 DCT の計算手順



●行ブロックごとの計算

$$\begin{cases} C_{0,0}=\dfrac{P_{0,0}+P_{0,1}}{2},\ C_{0,1}=\dfrac{P_{0,0}-P_{0,1}}{2} \\ C_{1,0}=\dfrac{P_{1,0}+P_{1,1}}{2},\ C_{1,1}=\dfrac{P_{1,0}-P_{1,1}}{2} \end{cases} \quad\cdots\cdots(14)$$

○ 式(7)～式(9)による2ステップ計算

●行ブロックごとの計算

$$\begin{cases} Q_{0,0}=\dfrac{s_{0,0}+s_{0,1}}{2},\ Q_{0,1}=\dfrac{s_{0,0}-s_{0,1}}{2} \\ Q_{1,0}=\dfrac{s_{1,0}+s_{1,1}}{2},\ Q_{1,1}=\dfrac{s_{1,0}-s_{1,1}}{2} \end{cases} \quad\cdots\cdots(15)$$

●列ブロックごとの計算

$$\begin{cases} C_{0,0}=\dfrac{Q_{0,0}+Q_{1,0}}{2},\ C_{0,1}=\dfrac{Q_{0,1}+Q_{1,1}}{2} \\ C_{1,0}=\dfrac{Q_{0,0}-Q_{1,0}}{2},\ C_{1,1}=\dfrac{Q_{0,1}-Q_{1,1}}{2} \end{cases} \quad\cdots\cdots(16)$$

● 2次元DCT値の計算結果

式(12)～式(16)に式(10)の画像データを代入して算出されるDCT値は以下のようになる．

○ 直接計算[式(12)]によるDCT値[図23.6(a)]

$$\begin{cases} C_{0,0}=\dfrac{14+2+4+0}{4}=5,\ C_{0,1}=\dfrac{14-2+4-0}{4}=4 \\ C_{1,0}=\dfrac{14+2-4-0}{4}=3,\ C_{1,1}=\dfrac{14-2-4+0}{4}=2 \end{cases} \quad\cdots(17)$$

○ 列[式(13)]から行[式(14)]の順で算出したDCT値[図23.6(b)]

●列ブロックごとの計算

$$\begin{cases} P_{0,0}=\dfrac{14+4}{2}=9,\ P_{0,1}=\dfrac{2+0}{2}=1 \\ P_{1,0}=\dfrac{14-4}{2}=5,\ P_{1,1}=\dfrac{2-0}{2}=1 \end{cases}$$

●行ブロックごとの計算

$$\begin{cases} C_{0,0}=\dfrac{9+1}{2}=5,\ C_{0,1}=\dfrac{9-1}{2}=4 \\ C_{1,0}=\dfrac{5+1}{2}=3,\ C_{1,1}=\dfrac{5-1}{2}=2 \end{cases} \quad\cdots\cdots(18)$$

○ 行[式(15)]から列[式(16)]の順で算出したDCT値[図23.6(c)]

●行ブロックごとの計算

$$\begin{cases} Q_{0,0}=\dfrac{14+2}{2}=8,\ Q_{0,1}=\dfrac{14-2}{2}=6 \\ Q_{1,0}=\dfrac{4+0}{2}=2,\ Q_{1,1}=\dfrac{4-0}{2}=2 \end{cases}$$

●列ブロックごとの計算

$$\begin{cases} C_{0,0}=\dfrac{8+2}{2}=5,\ C_{0,1}=\dfrac{6+2}{2}=4 \\ C_{1,0}=\dfrac{8-2}{2}=3,\ C_{1,1}=\dfrac{6-2}{2}=2 \end{cases} \quad\cdots\cdots(19)$$

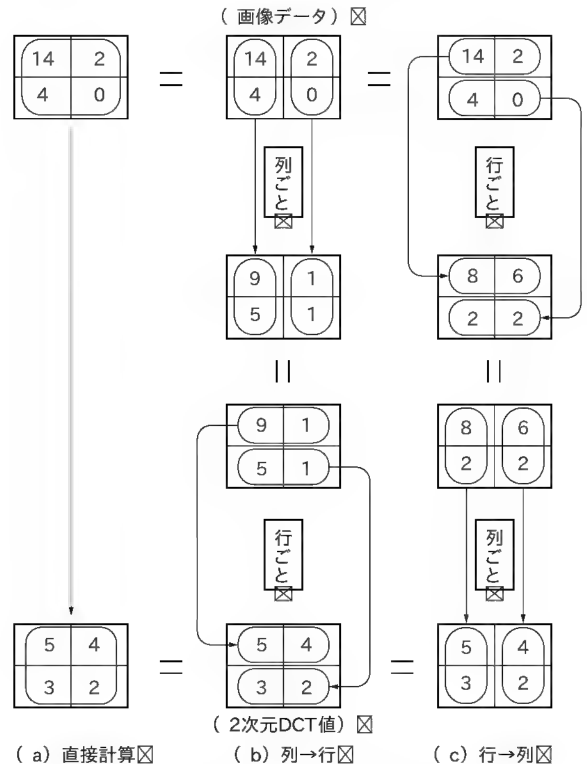


図23.6　2次元DCTの計算例(2×2画素の場合)

以上の結果[式(17)～式(19)]からも明らかなように，異なる計算手順に基づく2次元DCT値がすべて等しくなるわけで，式(2)～式(9)の2次元DCT計算式の妥当性が検証される．

2次元IDCTの定義と計算手順

逆に，2次元IDCTは，$m,n=0,1,2,\cdots,(N-1)$に対して，

$$s_{m,n}=\sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1}\gamma_k\gamma_\ell C_{k,\ell}\cos\left[\frac{(2m+1)k}{2N}\pi\right]\cos\left[\frac{(2n+1)\ell}{2N}\pi\right] \quad\cdots\cdots(20)$$

で表される．なお，式(20)の2次元DCT計算式は，式(2)を$N\times N$個の未知変数$\{s_{m,n}\}_{m,n=0}^{m,n=N-1}$に対する連立方程式とみなしたときの解に相当している．

それでは，2次元DCTの計算の場合と同様に，式(20)の2次元IDCT値が1次元IDCTの計算を2回適用して算出されることを示してみよう(読者のみなさんにもチャレンジしてもらいたい)．

式(20)の総和計算する処理(Σ)について，

$$s_{m,n}=\sum_{\ell=0}^{N-1}\gamma_\ell\cos\left[\frac{(2n+1)\ell}{2N}\pi\right]\left\{\sum_{k=0}^{N-1}\gamma_k C_{k,\ell}\cos\left[\frac{(2m+1)k}{2N}\pi\right]\right\} \quad\cdots\cdots(21)$$

と変形すれば，

$$\begin{cases} p_{m,\ell} = \displaystyle\sum_{k=0}^{N-1} \gamma_k C_{k,\ell} \cos\left[\frac{(2m+1)k}{2N}\pi\right] & \cdots\cdots\cdots\cdots\cdots (22) \\ s_{m,n} = \displaystyle\sum_{\ell=0}^{N-1} \gamma_\ell\, p_{m,\ell} \cos\left[\frac{(2n+1)\ell}{2N}\pi\right] & \cdots\cdots\cdots\cdots\cdots (23) \end{cases}$$

と表される．したがって，式(22)の列方向の1次元IDCTにより計算される値 $\{p_{m,\ell}\}_{m,\ell=0}^{m,\ell=N-1}$ に対して，さらに式(23)の行方向の1次元IDCTを適用すれば，2次元IDCT計算が実行されたことに等価であることが理解される［図23.7(a)］．

同様に，式(20)の総和計算する処理(Σ)について，

$$s_{m,n} = \sum_{k=0}^{N-1} \gamma_k \cos\left[\frac{(2m+1)k}{2N}\pi\right] \left\{\sum_{\ell=0}^{N-1} \gamma_\ell C_{k,\ell} \cos\left[\frac{(2n+1)\ell}{2N}\pi\right]\right\} \cdots\cdots\cdots (24)$$

と変形すれば，

$$\begin{cases} q_{k,n} = \displaystyle\sum_{\ell=0}^{N-1} \gamma_\ell C_{k,\ell} \cos\left[\frac{(2n+1)\ell}{2N}\pi\right] & \cdots\cdots\cdots\cdots\cdots (25) \\ s_{m,n} = \displaystyle\sum_{k=0}^{N-1} \gamma_k q_{k,n} \cos\left[\frac{(2m+1)k}{2N}\pi\right] & \cdots\cdots\cdots\cdots\cdots (26) \end{cases}$$

と表される．よって，式(25)の行方向の1次元IDCTにより計算される値 $\{q_{k,n}\}_{k,n=0}^{k,n=N-1}$ に対して，さらに式(26)の列方向の1次元IDCTを適用すれば，2次元IDCT計算が実行されるのである［図23.7(b)］．

例題1

いま，2×2個の2次元DCT値 $\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=1}$ が，

$$\begin{bmatrix} C_{0,0} & C_{0,1} \\ C_{1,0} & C_{1,1} \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 3 & 2 \end{bmatrix} \cdots\cdots\cdots\cdots\cdots (27)$$

であるとき，画像の2次元画素データ，すなわち，

$$\begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix} \cdots\cdots\cdots\cdots\cdots (28)$$

を求めよ．そのとき，式(20)の直接計算による結果が，式(21)～式(26)に基づく算出結果に一致することを確認せよ．

解答1

N=2に対する3通りの計算式を適用してIDCT値を求め，すべて等しくなることを検証すればよい．

● 2次元IDCT計算式

式(20)～式(26)において，N=2を代入することにより得られる2次元IDCTの3通りの計算式を以下に示す．

○ 式(20)による直接計算

$$\begin{cases} s_{0,0} = C_{0,0} + C_{0,1} + C_{1,0} + C_{1,1} \\ s_{0,1} = C_{0,0} - C_{0,1} + C_{1,0} - C_{1,1} \\ s_{1,0} = C_{0,0} + C_{0,1} - C_{1,0} - C_{1,1} \\ s_{1,1} = C_{0,0} - C_{0,1} - C_{1,0} + C_{1,1} \end{cases} \cdots\cdots\cdots\cdots\cdots (29)$$

○ 式(21)～式(23)による2ステップ計算

まず，列(縦)方向の二つのDCT値を1ブロックとし，1次元IDCT値を求める．この得られた1次元IDCT値の行(横)方向の二つをまとめて1ブロックとし，1次元IDCT値を求める．

● 列ブロックごとの計算

$$\begin{cases} p_{0,0} = C_{0,0} + C_{1,0},\ p_{0,1} = C_{0,1} + C_{1,1} \\ p_{1,0} = C_{0,0} - C_{1,0},\ p_{1,1} = C_{0,1} - C_{1,1} \end{cases} \cdots\cdots\cdots (30)$$

● 行ブロックごとの計算

$$\begin{cases} s_{0,0} = p_{0,0} + p_{0,1},\ s_{0,1} = p_{0,0} - p_{0,1} \\ s_{1,0} = p_{1,0} + p_{1,1},\ s_{1,1} = p_{1,0} - p_{1,1} \end{cases} \cdots\cdots\cdots (31)$$

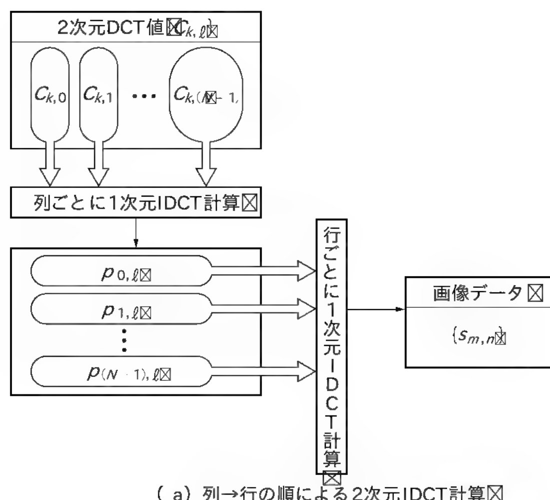
○ 式(24)～式(26)による2ステップ計算

● 行ブロックごとの計算

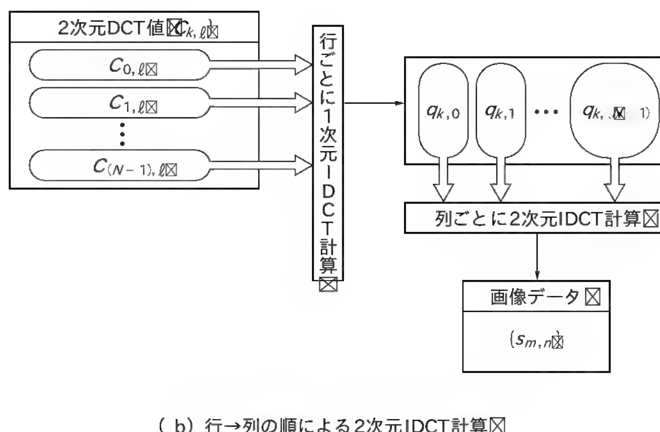
$$\begin{cases} q_{0,0} = C_{0,0} + C_{0,1},\ q_{0,1} = C_{0,0} - C_{0,1} \\ q_{1,0} = C_{1,0} + C_{1,1},\ q_{1,1} = C_{1,0} - C_{1,1} \end{cases} \cdots\cdots\cdots (32)$$

● 列ブロックごとの計算

$$\begin{cases} s_{0,0} = q_{0,0} + q_{1,0},\ s_{0,1} = q_{0,1} + q_{1,1} \\ s_{1,0} = q_{0,0} - q_{1,0},\ s_{1,1} = q_{0,1} - q_{1,1} \end{cases} \cdots\cdots\cdots (33)$$



(a) 列→行の順による2次元IDCT計算



(b) 行→列の順による2次元IDCT計算

図23.7 1次元IDCTを適用した2次元IDCTの計算手順



● 2次元IDCT値の計算結果

式(29)～式(33)に式(27)のDCT値を代入して算出される画像データは以下のようになる．

○ 直接計算［式(29)］によるIDCT値［図23.8(a)］

$$\begin{cases} s_{0,0}=5+4+3+2=14 \\ s_{0,1}=5-4+3-2=2 \\ s_{1,0}=5+4-3-2=4 \\ s_{1,1}=5-4-3+2=0 \end{cases} \quad \cdots\cdots (34)$$

○ 列［式(30)］から行［式(31)］の順で算出したDCT値［図23.8(b)］

● 列ブロックごとの計算

$$\begin{cases} p_{0,0}=5+3=8,\ p_{0,1}=4+2=6 \\ p_{1,0}=5-3=2,\ p_{1,1}=4-2=2 \end{cases}$$

● 行ブロックごとの計算

$$\begin{cases} s_{0,0}=8+6=14,\ s_{0,1}=8-6=2 \\ s_{1,0}=2+2=4,\ s_{1,1}=2-2=0 \end{cases} \quad \cdots\cdots (35)$$

○ 行［式(32)］から列［式(33)］の順で算出したIDCT値（図23.8(c)］

● 行ブロックごとの計算

$$\begin{cases} q_{0,0}=5+4=9,\ q_{0,1}=5-4=1 \\ q_{1,0}=3+2=5,\ q_{1,1}=3-2=1 \end{cases}$$

● 列ブロックごとの計算

$$\begin{cases} s_{0,0}=9+5=14,\ s_{0,1}=1+1=2 \\ s_{1,0}=9-5=4,\ s_{1,1}=1-1=0 \end{cases} \quad \cdots\cdots (36)$$

以上の結果［式(34)～式(36)］からも明らかなように，異なる計算手順に基づく2次元IDCT値がすべて等しくなるわけで，式(20)～式(26)の2次元IDCT計算式の妥当性が検証される．

2次元DCTによる画像の直交展開

ここでは，2次元DCTが意味する物理的な考えかたを解説する．まず，$m,n=0,1,2,\cdots,(N-1)$に対して，

$$\lambda_{m,n}^{(k,\ell)}=\gamma_k\gamma_\ell\cos\left[\frac{(2m+1)k}{2N}\pi\right]\cos\left[\frac{(2n+1)\ell}{2N}\pi\right] \quad \cdots\cdots (37)$$

と定義して，$\{\lambda_{m,n}^{(k,\ell)}\}_{m,n=0}^{m,n=N-1}$を$\boldsymbol{\lambda}^{(k,\ell)}$，画像データ$\{s_{m,n}\}_{m,n=0}^{m,n=N-1}$を$S$と表せば，式(2)の2次元DCTは，

$$S=\sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1}C_{k,\ell}\,\boldsymbol{\lambda}^{(k,\ell)} \quad \cdots\cdots (38)$$

と簡略表現される．式(38)の表現は画像データの正規直交基底画像による展開，そして2次元DCT値$\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$は展開係数とよばれる．なお，正規直交基底画像とは，

$$\frac{1}{N^2}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\lambda_{m,n}^{(k,\ell)}\lambda_{m,n}^{(i,j)}=\begin{cases}1 & (k=i,\ \ell=j)\\ 0 & (k\neq i,\ \text{または}\ \ell\neq j)\end{cases} \quad \cdots (39)$$

となる関係が成立するものをいう．ただ式(37)～式(39)は少々わかりにくいと思われるので，例題1 の2×2画素の画像データに対する2次元DCT値を用いて具体的に説明してみよう．

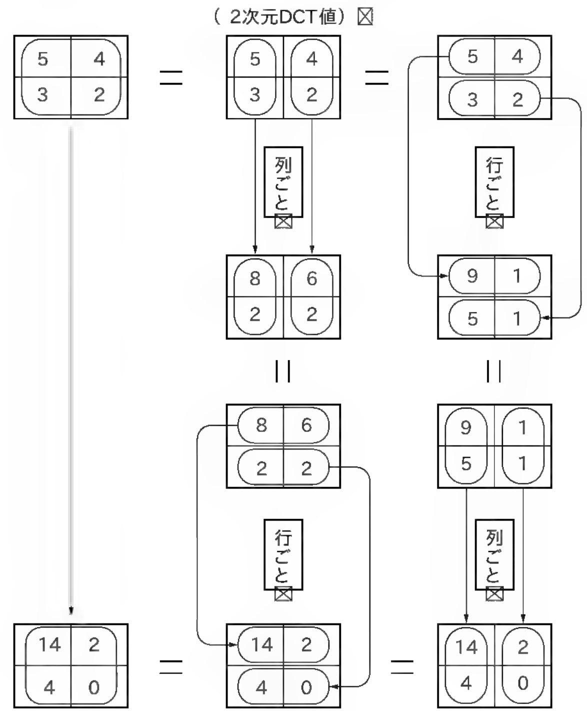


図23.8 2次元IDCTの計算例(2×2画素の場合)

まず，画像データ$\{s_{m,n}\}_{m,n=0}^{m,n=1}$［式(34)］と2次元DCT値$\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=1}$［式(27)］をそれぞれ再掲しておく．

$$\begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix}=\begin{bmatrix} 14 & 2 \\ 4 & 0 \end{bmatrix} \quad \cdots\cdots (40)$$

$$\begin{bmatrix} C_{0,0} & C_{0,1} \\ C_{1,0} & C_{1,1} \end{bmatrix}=\begin{bmatrix} 5 & 4 \\ 3 & 2 \end{bmatrix} \quad \cdots\cdots (41)$$

ここで，式(37)より，

$$\begin{cases} \boldsymbol{\lambda}^{(0,0)}=\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},\ \boldsymbol{\lambda}^{(0,1)}=\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \\ \boldsymbol{\lambda}^{(1,0)}=\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix},\ \boldsymbol{\lambda}^{(1,1)}=\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \end{cases} \quad \cdots\cdots (42)$$

であり，式(38)に基づき，

$$S=C_{0,0}\boldsymbol{\lambda}^{(0,0)}+C_{0,1}\boldsymbol{\lambda}^{(0,1)}+C_{1,0}\boldsymbol{\lambda}^{(1,0)}+C_{1,1}\boldsymbol{\lambda}^{(1,1)} \quad \cdots\cdots (43)$$

となる関係が導かれる．よって，式(40)～式(42)を式(43)に代入すれば，

$$\begin{bmatrix} 14 & 2 \\ 4 & 0 \end{bmatrix}=5\times\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}+4\times\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}+3\times\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}+2\times\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

となる．

例題2

式(42)の各行列$\{\boldsymbol{\lambda}^{(k,\ell)}\}_{k,\ell=0}^{k,\ell=1}$が，式(39)を満たすことを示せ．

解答2

たとえば，$\boldsymbol{\lambda}^{(0,0)}$と$\boldsymbol{\lambda}^{(0,0)}$では，

$$\frac{1}{2^2}\sum_{m=0}^{1}\sum_{n=0}^{1}\lambda_{m,n}^{(0,0)}\lambda_{m,n}^{(0,0)}$$
$$=\frac{1}{4}\left(\lambda_{0,0}^{(0,0)}\lambda_{0,0}^{(0,0)}+\lambda_{0,1}^{(0,0)}\lambda_{0,1}^{(0,0)}+\lambda_{1,0}^{(0,0)}\lambda_{1,0}^{(0,0)}+\lambda_{1,1}^{(0,0)}\lambda_{1,1}^{(0,0)}\right)$$
$$=\frac{1}{4}(1\times1+1\times1+1\times1+1\times1)=1 \quad \cdots\cdots(44)$$

となる．また，$\boldsymbol{\lambda}^{(1,0)}$と$\boldsymbol{\lambda}^{(0,1)}$では，

$$\frac{1}{2^2}\sum_{m=0}^{1}\sum_{n=0}^{1}\lambda_{m,n}^{(1,0)}\lambda_{m,n}^{(0,1)}$$
$$=\frac{1}{4}\left(\lambda_{0,0}^{(1,0)}\lambda_{0,0}^{(0,1)}+\lambda_{0,1}^{(1,0)}\lambda_{0,1}^{(0,1)}+\lambda_{1,0}^{(1,0)}\lambda_{1,0}^{(0,1)}+\lambda_{1,1}^{(1,0)}\lambda_{1,1}^{(0,1)}\right)$$
$$=\frac{1}{4}\{1\times1+1\times(-1)+(-1)\times1+(-1)\times(-1)\}$$
$$=\frac{1}{4}(1-1-1+1)=0 \quad \cdots\cdots(45)$$

となる．ほかの計算は省略するが，式(39)の正規直交基底画像の条件が成立することを各自検証しておいてもらいたい．

以上の結果に基づき，2次元DCTの物理的な意味を考えてみることにする．最初は正規直交基底画像$\{\boldsymbol{\lambda}^{(k,\ell)}\}_{k,\ell=0}^{k,\ell=1}$からで，各基底画像を見やすくするために，

1 → 白，(－1) → 黒

に対応させると図23.9が得られる．図(a)は直流画像，図(b)は縦じま模様，図(c)は横じま模様，図(d)は格子じま模様であり，正規直交基底画像$\{\boldsymbol{\lambda}^{(k,\ell)}\}_{k,\ell=0}^{k,\ell=1}$の上付き文字について，

$$\begin{cases} k\text{は横じまの個数} \\ \ell\text{は縦じまの個数} \end{cases} \quad \cdots\cdots(46)$$

であることがわかる．したがって，式(42)と式(43)に基づき，2次元DCT値$\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=1}$の下付き文字については，

$$\begin{cases} k\text{は横じまの個数} \\ \ell\text{は縦じまの個数} \end{cases} \quad \cdots\cdots(47)$$

を表すことも理解される．なお，式(46)と式(47)は一般的に$N\times N$画素の画像データに対しても成立する．

このように，画像データの2次元DCT値$\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=1}$は正規直交基底画像$\boldsymbol{\lambda}^{(k,\ell)}=\{\lambda_{m,n}^{(k,\ell)}\}_{m,n=0}^{m,n=N-1}$の含まれる割合を与えるものであることもわかる．言い換えれば，2次元DCTは画像データの正規直交基底画像による分解算法，すなわち画像の周波数スペクトルの分析手法なのである．

雑音を除去する処理

いま，雑音を含む画像信号$\{s_{m,n}\}_{m,n=0}^{m,n=N-1}$，

$$s_{m,n}=x_{m,n}+n_{m,n} \quad \cdots\cdots(48)$$

ただし，$x_{m,n}$：信号成分，$n_{m,n}$：雑音成分

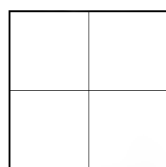
から信号成分を取り出す処理において，2次元DCT計算を利用することを考えてみよう．

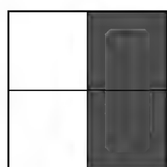
基本的には，雑音を含む信号$\{s_{m,n}\}_{m,n=0}^{m,n=N-1}$を2次元DCTして周波数成分を分析し，その周波数スペクトル値$\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$に対して，信号成分と思われるものには"1"，雑音成分と思われるものには"0"を掛けて，さらに2次元IDCTするのである．つまり，信号は雑音よりスペクトル値が大きいことを利用し，信号の周波数スペクトルには"1"を掛けて取り出し，不要な雑音には"0"を掛けて取り除くという構図である(図23.10)．

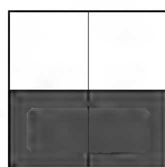
以下に，雑音除去の処理手順について，具体的に示しておこう．

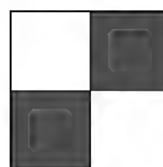
手順1　2次元DCTによる周波数成分の計算

雑音を含む信号$\{s_{m,n}\}_{m,n=0}^{m,n=N-1}$を，式(2)に基づき，2次元DCT


(a) 直流画像$\boldsymbol{\lambda}^{(0,0)}$


(b) 縦じま模様$\boldsymbol{\lambda}^{(0,1)}$


(c) 横じま模様$\boldsymbol{\lambda}^{(1,0)}$


(c) 格子じま模様$\boldsymbol{\lambda}^{(1,1)}$

（□は1，■は(－1)に対応）

図23.9　正規直交基底画像(2×2画素の場合)

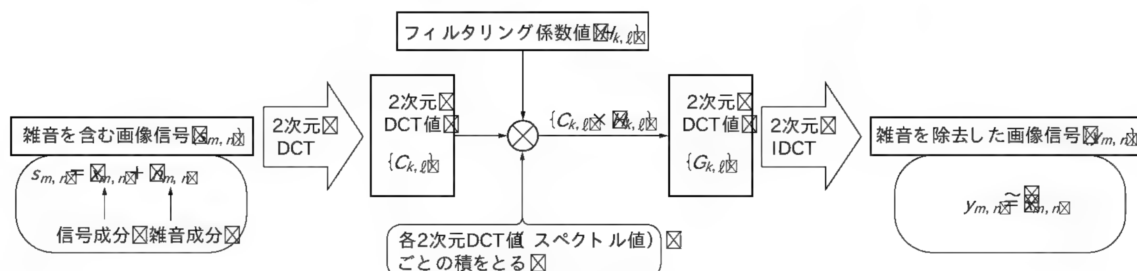


図23.10　2次元DCT，IDCTによる雑音の除去(フィルタリング)プロセス



計算し，その周波数成分 $\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$ を求める．

手順2　信号と雑音の識別判定

周波数成分 $\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$ の値の大小を見て，信号と雑音の識別を行う．たとえば，

$$\begin{cases} |C_{k,\ell}| \geq \varepsilon \text{であれば，} C_{k,\ell} \text{は信号成分} \\ |C_{k,\ell}| < \varepsilon \text{であれば，} C_{k,\ell} \text{は雑音成分} \end{cases} \quad \cdots\cdots (49)$$

とすればよい．ここで，ε は雑音と信号とを切り分けるための判定レベル(しきい値)であり，適切に定めておく必要がある．

手順3　雑音除去の計算

周波数成分 $\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$ に掛ける係数を $\{H_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$ とするとき，

$$\begin{cases} \cdot \text{ 信号成分に対しては } H_{k,\ell} = 1 \\ \cdot \text{ 雑音成分に対しては } H_{k,\ell} = 0 \end{cases} \quad \cdots\cdots (50)$$

として，

$$G_{k,\ell} = C_{k,\ell} \times H_{k,\ell} \quad \cdots\cdots (51)$$

を計算して，雑音を除去した周波数スペクトル特性 $\{G_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$ を作成する．

手順4　2次元IDCTによる時間波形の再合成

手順3で得られた信号のみの周波数成分 $\{G_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$ を有する信号 $\{y_{m,n}\}_{m,n=0}^{m,n=N-1}$ を再合成するために，

$$y_{m,n} = \sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1} \gamma_k \gamma_\ell G_{k,\ell} \cos\left[\frac{(2m+1)k}{2N}\pi\right] \cos\left[\frac{(2n+1)\ell}{2N}\pi\right] \quad \cdots\cdots (52)$$

に基づき，2次元IDCT計算する．

以上の手順を踏んで，2次元のDCT値とIDCT値を算出することにより，ディジタル雑音除去システムが実現できるという流れである．

例題3

図23.11に示す雑音を含んだ画像信号から，2次元のDCTとIDCTを利用して雑音を除去したい．手順1～手順4に基づき，フィルタリング処理するようすについて信号値を計算し，確認せよ．ただし，信号と雑音を識別する判定レベル ε は10とする．

解答3

以下に，手順ごとに信号値の計算結果を示す(図23.12)．

手順1　周波数成分(2次元DCT値)の計算

信号値($s_{0,0}=107$，$s_{0,1}=-111$，$s_{1,0}=-83$，$s_{1,1}=99$)を式(12)に代入し，2次元DCTして周波数成分を求める．

$$\begin{cases} C_{0,0} = \dfrac{107+(-111)+(-83)+99}{4} = 3 \\ C_{0,1} = \dfrac{107-(-111)+(-83)-99}{4} = 9 \\ C_{1,0} = \dfrac{107+(-111)-(-83)-99}{4} = -5 \\ C_{1,1} = \dfrac{107-(-111)-(-83)+99}{4} = 100 \end{cases} \quad \cdots\cdots (53)$$

107	(−111)
(−83)	99

図23.11 **例題3** 雑音を含む画像データ

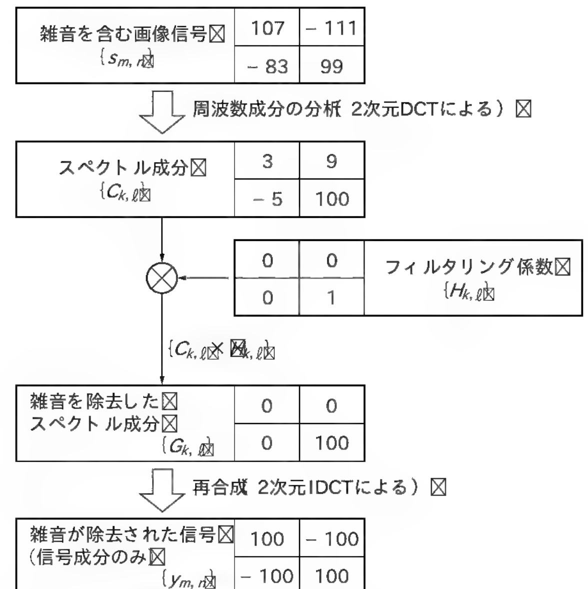


図23.12 **例題3** のフィルタリング処理

手順2　信号と雑音の識別判定

手順1の結果から，判定レベル $\varepsilon=10$ なので，式(49)に基づき，信号と雑音を切り分ける．

$$\begin{cases} |C_{0,0}| = 3 < 10 \Rightarrow \text{雑音成分} \\ |C_{0,1}| = 9 < 10 \Rightarrow \text{雑音成分} \\ |C_{1,0}| = 5 < 10 \Rightarrow \text{雑音成分} \\ |C_{1,1}| = 100 \geq 10 \Rightarrow \text{信号成分} \end{cases} \quad \cdots\cdots (54)$$

手順3　雑音除去の計算

各周波数成分 $\{C_{0,0}, C_{0,1}, C_{1,0}, C_{1,1}\}$ ごとに掛けるフィルタリング係数値は，

$$\begin{cases} H_{0,0} = 0 \text{(雑音なので，除去する)} \\ H_{0,1} = 0 \text{(雑音なので，除去する)} \\ H_{1,0} = 0 \text{(雑音なので，除去する)} \\ H_{1,1} = 1 \text{(信号なので，通す)} \end{cases}$$

とすればよい．その結果，雑音が取り除かれた信号の周波数成分は以下のようになる．

$$\begin{bmatrix} G_{0,0} & G_{0,1} \\ G_{1,0} & G_{1,1} \end{bmatrix} = \begin{bmatrix} C_{0,0} \times H_{0,0} & C_{0,1} \times H_{0,1} \\ C_{1,0} \times H_{1,0} & C_{1,1} \times H_{1,1} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 100 \end{bmatrix} \quad \cdots\cdots (55)$$

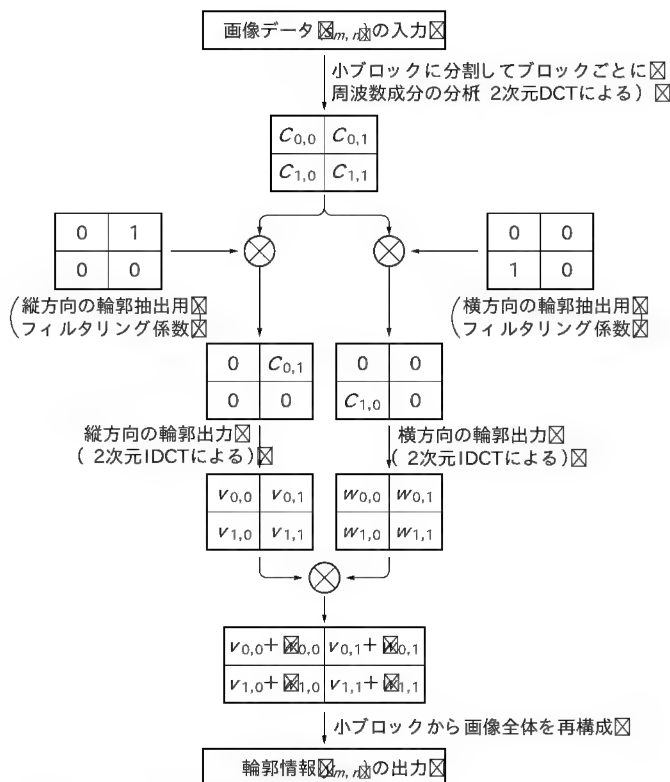
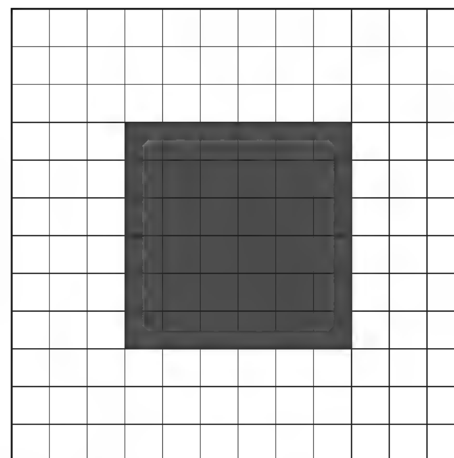


図23.13 輪郭(エッジ)を抽出する処理プロセス例



[□は2, ■は(−2)とする]

図23.14 例題4 の画像データ

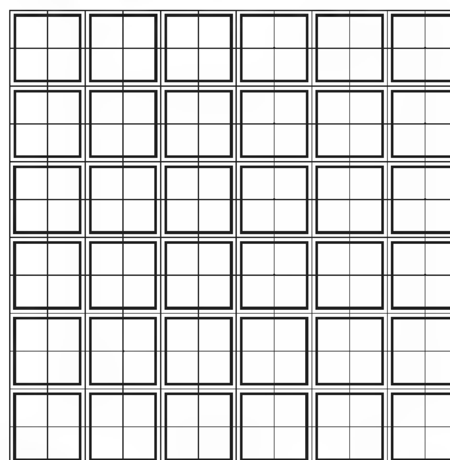


図23.15 画像処理における小ブロック化の例(2×2画素の場合)

手順4 2次元IDCTによる時間波形の再合成

手順3で得られた周波数成分$\{G_{0,0}, G_{0,1}, G_{1,0}, G_{1,1}\}$を有する信号を再合成するために，式(52)に基づき，N=2として2次元IDCT値を計算する．すなわち，

$$\begin{cases} y_{0,0}=G_{0,0}+G_{0,1}+G_{1,0}+G_{1,1} \\ y_{0,1}=G_{0,0}-G_{0,1}+G_{1,0}-G_{1,1} \\ y_{1,0}=G_{0,0}+G_{0,1}-G_{1,0}-G_{1,1} \\ y_{1,1}=G_{0,0}-G_{0,1}-G_{1,0}+G_{1,1} \end{cases} \quad \cdots\cdots\cdots (56)$$

となる関係より，式(55)の値$\{G_{0,0}, G_{0,1}, G_{1,0}, G_{1,1}\}=\{0, 0, 0, 100\}$を式(56)に代入し，出力信号を計算する．

$$\begin{cases} y_{0,0}=0+0+0+100=100 \\ y_{0,1}=0-0+0-100=-100 \\ y_{1,0}=0+0-0-100=-100 \\ y_{1,1}=0-0-0+100=100 \end{cases} \quad \cdots\cdots\cdots (57)$$

以上より，$y_{m,n}=x_{m,n}$であることから，雑音を除去できたことになる．

輪郭(エッジ)を抽出する処理

ところで，文字画像の認識をはじめとして，顔や医用診断画像の特徴抽出などの画像処理においては，文字の輪郭や病変部分を切り出す際には，必ずと言ってよいほど輪郭情報が必要になってくる．そこで，こうした輪郭情報を得る処理(エッジ抽出処理ともいう)の2次元DCTバージョンを作ってみよう．基本的なしくみは前述の雑音除去の考えかたを拡張するだけで簡単にできてしまう(図23.13)．

手順1 2次元DCTによる周波数成分の計算

式(2)に基づき，画像データ$\{s_{m,n}\}_{m,n=0}^{m,n=N-1}$を2次元DCT計算して，その周波数成分$\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$を求める．

手順2 輪郭抽出のための乗算係数の調整

周波数成分$\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$に掛ける乗算係数を$\{H_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$とするとき，

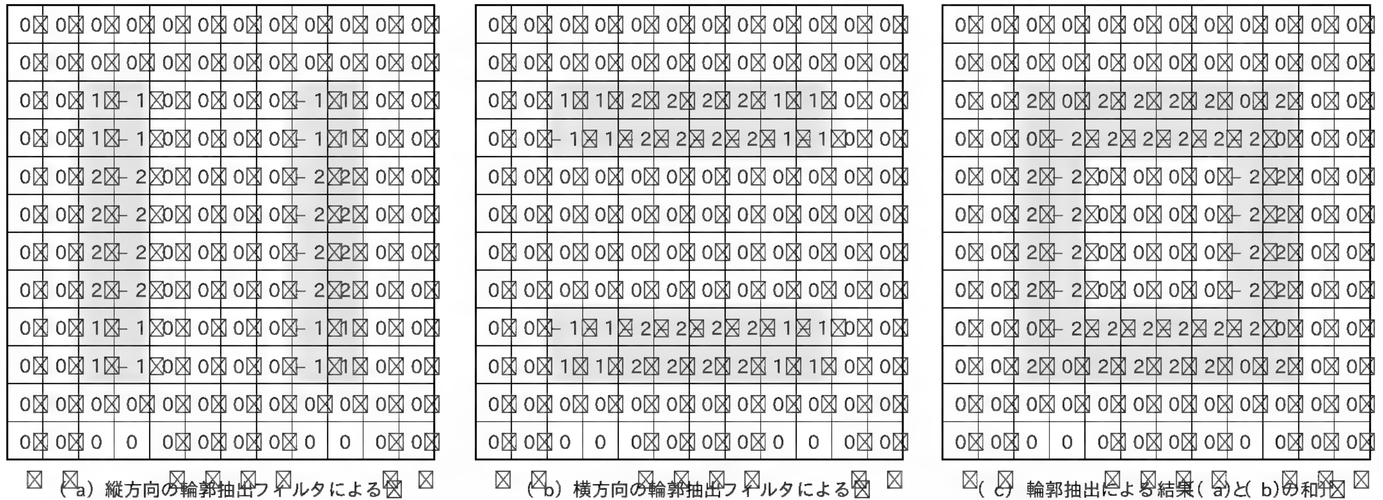
- 輪郭情報に該当する周波数成分(k, ℓ)に対しては，$H_{k,\ell}$を適切な大きい値
- 輪郭情報に該当しない周波数成分(k, ℓ)に対しては，$H_{k,\ell}$を適切な小さい値

に設定し，

$$G_{k,\ell}=C_{k,\ell}\times H_{k,\ell} \quad \cdots\cdots\cdots (58)$$

を計算して，輪郭情報を取り出した周波数スペクトル特性$\{G_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$を作成する．このとき，$\{H_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$が輪郭抽出





0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	-1	0	0	0	0	-1	1	0	0
0	0	1	-1	0	0	0	0	-1	1	0	0
0	0	2	-2	0	0	0	0	-2	2	0	0
0	0	2	-2	0	0	0	0	-2	2	0	0
0	0	2	-2	0	0	0	0	-2	2	0	0
0	0	2	-2	0	0	0	0	-2	2	0	0
0	0	1	-1	0	0	0	0	-1	1	0	0
0	0	1	-1	0	0	0	0	-1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

(a) 縦方向の輪郭抽出フィルタによる

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	2	2	2	2	1	1	0	0
0	0	-1	-1	-2	-2	-2	-2	-1	-1	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	-1	-1	-2	-2	-2	-2	-1	-1	0	0
0	0	1	1	2	2	2	2	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

(b) 横方向の輪郭抽出フィルタによる

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	2	0	2	2	2	2	0	2	0	0
0	0	0	-2	-2	-2	-2	-2	-2	0	0	0
0	0	2	-2	0	0	0	0	-2	2	0	0
0	0	2	-2	0	0	0	0	-2	2	0	0
0	0	2	-2	0	0	0	0	-2	2	0	0
0	0	2	-2	0	0	0	0	-2	2	0	0
0	0	0	-2	-2	-2	-2	-2	-2	0	0	0
0	0	2	0	2	2	2	2	0	2	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

(c) 輪郭抽出による結果((a)と(b)の和)

図 23.16 例題4 の処理結果

用の係数パラメータになる．

手順3 2次元IDCTによる輪郭情報の出力

手順2で得られた周波数成分$\{G_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$を有する信号を，式(52)に基づき，輪郭情報を合成して出力する．

例題4

いま，図23.14に示す画像データ(正方形)の輪郭情報を取り出したい．そこで，図23.14の画像データを2×2画素の小ブロックに分割し，ブロックごとの周波数成分を分析して処理することを考える．**手順1**～**手順3**に基づき，2次元のDCT，IDCT計算を利用して処理した後の信号値を示せ．

解答4

まず，図23.14の画像データを図23.15のようにブロック化する．**例題3**の雑音除去の例とほぼ同じ処理であり，**手順3**の輪郭抽出用の係数パラメータ$\{H_{k,\ell}\}_{k,\ell=0}^{k,\ell=1}$を，

- 縦方向の輪郭抽出用

$$H_{0,0}=0,\ H_{0,1}=1,\ H_{1,0}=0,\ H_{1,1}=0 \cdots\cdots (59)$$

- 横方向の輪郭抽出用

$$H_{0,0}=0,\ H_{0,1}=0,\ H_{1,0}=1,\ H_{1,1}=0 \cdots\cdots (60)$$

と設定することにより，輪郭抽出を2次元のDCT，IDCT計算で実現できる．図23.16に，計算結果のみを示しておくので，必ず検算してもらいたい．

* *

以上で数回にわたるDCT(ディジタル・コサイン変換)の説明を終わり，次回からは本連載の総まとめに入る．お楽しみに．

みたに・まさあき　東京電機大学工学部情報通信工学科

オープン・ソース・ソフトウェア
DSP Gatewayを用いた

Linuxで使うOMAP DSP部のソフトウェア開発環境



森 英悟/小林 俊浩/高橋 清隆

はじめに

「DSP Gateway」は，米国Texas Instruments社(以下，TI社)のOMAPという，ARMコアとDSPコアを内蔵したデュアルコア・プロセッサのDSP部を，ARM上のLinuxアプリケーションから使う手段を提供するソフトウェアです．DSP Gatewayには，Linuxデバイス・ドライバ，DSP側のライブラリ，それにコントロール・ユーティリティが含まれています．

本ソフトウェアは，オープン・ソース・ソフトウェアとして開発が進められています．ライセンスにはGPLを採用しています．すべてのソースおよびドキュメントは，

http://dspgateway.sourceforge.net/

からダウンロードすることができます．

1. OMAP5910/1510とLinux

● OMAP5910/1510の構成

OMAP5910およびOMAP1510プロセッサは，携帯電話やPDAなどのモバイル端末をターゲットとした低消費電力のプロセッサです．CPUコアにメモリ・コントローラやシリアル，USB，MMC/SDカードなどの各種コントローラなどが集積されています．もっとも特徴的なのは，ARM(ARM925)コアとDSP(TMS320C55x)コアの二つのコアを備えているという点です．



図1 カーネルのコンフィグレーション・メニュー
DSP Gatewayパッチを当てることにより，コンフィグレーション・メニューに「DSP driver」の項目が追加される．

● OMAPで動作するOS

OMAP上で動作するOSとして，Windows CE，Symbian，Palm OS，そしてLinuxなどがあります．これらのOSはいずれもARMコア上で動作し，DSPをコプロセッサとして使用します．

● Linux環境

Linuxには，ARM Linuxを用います．ARM LinuxをOMAP上で動作させるためのパッチを米国MontaVista Software社が公開しています．パッチは次に示すFTPサイトから入手できます．

ftp://source.mvista.com/pub/omap/2.4/

このカーネルは，OMAPプラットホーム用開発キットであるInnovator上で動作します．しかし，これにはDSPのサポートが含まれていないため，DSPを使うためには別途ドライバが必要になります．そこで登場するのがDSP Gatewayです．

● DSP Gatewayを組み込む

MontaVista Software社のOMAPパッチを適用したカーネルに，DSP Gatewayパッチを当てます．すると図1のようにカーネルのコンフィグレーション・メニューの“System Type”→“TI OMAP Implementations”に“OMAP1510 DSP driver”の項目が追加されます．これにチェックを入れることにより，DSP Gatewayがカーネルに組み込まれます．DSP Gatewayをカーネル・モジュールにすることも可能です．

2. DSP Gatewayのコンセプト

DSP Gatewayは，アプリケーション・プログラマがDSPをもっと手軽に利用できるしくみを提供することを目的としています．一般にDSPは音声や画像のCODECなどに使用されていますが，たとえば「ARM側からMPEGデータを送ればデコードして返してくれる便利な箱」というような単純な使いかたができるようにと考えて開発されています(図2)．

もう少し説明すると，DSPの機能をデバイス・ファイルとして見せます．Linuxのアプリケーション・プログラムは，このデバイス・ファイルに対してデータを読み書きすることによって，DSPタスクと通信を行います．

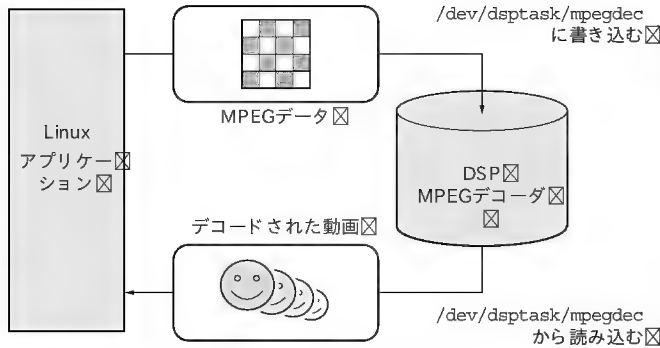


図2　DSP Gatewayのコンセプト
単純なデバイス・ファイル・インターフェースを採用し，簡単に使えるように考えられている．

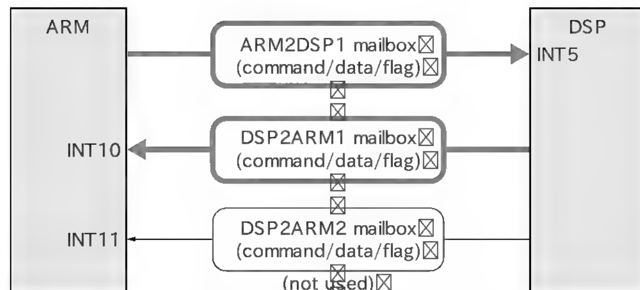


図3　mailboxのしくみ
割り込みを使ってコマンドおよびデータをARMとDSP間で双方向に通信することができる．

3. DSP Gatewayのアーキテクチャ

次にDSP Gatewayのアーキテクチャの概要を説明します．詳細については，DSP Gatewayの仕様書(1)を参照してください．

● mailbox――ARMとDSP間の通信手段

OMAPプロセッサには，ARMとDSP間の通信手段としてmailboxというしくみが用意されています．1本のmailboxは，16ビットのコマンドと16ビットのデータ，および1ビットのフラグのレジスタで構成されています．このmailboxは，ARMからDSP方向に1本，DSPからARM方向に2本あります．送信側のプロセッサがこのレジスタにデータを書き込むことにより，受信側のプロセッサに割り込みが入り，レジスタの内容を伝達することができます．DSP Gatewayでは，各方向1本ずつのmailboxを使用します(図3)．

DSP Gatewayにおいては，コマンド・レジスタの上位8ビットを用いて`Mailbox`コマンドを定義しています(下位8ビットには，通信対象となるDSPタスクのIDなどの付加的な情報が入る)．おもな`Mailbox`コマンドとしては，表1のようなものがあります．

● 通信バッファ――二つのプロセッサからアクセスできるメモリ空間

`Mailbox`コマンド，およびデータ・レジスタだけではプロセッサ間で大量のデータを通信することは不可能です．そのため，二つのプロセッサからアクセスできるメモリ空間に通信用のバッファを定義し，これを通じてブロック・データの送受信を行います．このバッファのことをIPBUF(Inter-Processor Buffer)と呼びます．IPBUFには，Global IPBUFとPrivate IPBUFの2種類があります．Global IPBUFは，複数のDSPタスクが共通に利用するもので，使用する前後でバッファの予約と解放が必要です．これとは逆にPrivate IPBUFは，各DSPタスクが自分専用に持つことができるもので，予約や解放の処理は必要ありません(ただし，一つのブロックを送信後，送信相手のプロセッサがそのデータを受け取るまで次のデータを送信できない)．IPBUFはDSPの内蔵メモリであるDARAMまたはSARAM，あるいはDSP空間にマップした外部メモリ(SDRAM)のどちらに配置することも可能です．

Global IPBUFは，固定サイズのブロックを複数個定義でき，それぞれのブロックをライン(line)と呼びます．各ラインは，BID(Buffer ID)と呼ぶ固有のIDによって区別されます．DSPタスクがGlobal IPBUFを使用する際には，まずtokliBIOS(後述するDSP GatewayのDSP側ライブラリ)のAPIを通して使用されていないIPBUFラインを予約し，そこにデータを書き込みます．また，Global IPBUFを使用してデータを受信し，バッファの使用が終了した際には同様にtokliBIOSのAPIを通してこのラインを解放しなくてはなりません．これらの予約と解放処理は，Linux側ではドライバ内部で行われるため，アプリケーションで意識する必要はないのですが，DSP側ではアプリケーション(タスク)が直接行うという点に注意してくださ

表1　おもなMailboxコマンド

コマンド	機　能
WDSND	Word Send 1ワード(16ビット)データを送信
WDREQ	Word Request 1ワード・データを要求
BKSND	Block Send Global IPBUFを使用してブロック・データを送信
BKREQ	Block Request Global IPBUFを使用してブロック・データを要求
BKYLD	Block Yield IPBUFラインのオーナ権を相手のプロセッサに譲る
BKSNDP	Block Send Private Private IPBUFを使用してブロック・データを送信
BKREQP	Block Request Private Private IPBUFを使用してブロック・データを要求
TCTL	Task Control DSPタスクにコントロール・コマンドを送信
TCFG	Task Configuration DSPタスクにコンフィグレーションを行う
DSPCFG	DSP Configuration DSPアプリケーション全体のコンフィグレーションを行う
ERR	Error エラー通知
DBG	Debug

い．Global IPBUFの動作例を図4に示します．

●メモリ・マッピング――ARMとDSPのアドレス空間

ARMとDSPは独立のアドレス空間を持ち，ARMは4Gバイト（32ビット・アドレス），DSPは16Mバイト（24ビット・アドレス）のメモリ空間を持ちます（DSPはメモリ空間とI/O空間が別だが，本稿ではI/O空間については省略する）．

まず先に，DSPのメモリ空間から説明します．OMAPにはDSP用のメモリとしてDARAMおよびSARAMと呼ぶRAMがオンチップで搭載されており，サイズはそれぞれ64Kバイトと96Kバイトです．さらにPDROMと呼ぶROMも内蔵されていますが，DSP Gatewayでは使用しません．また，OMAPにはDSP MMUが備えられていて，これを通してDSP空間に外部SDRAMをマッピングすることができます〔図5の（1）〕．マッピングなどのDSP MMUの設定は，ARMから行います．DSP Gatewayでは，この機能を使用してPDROMの領域に外部SDRAMをマッピングし，オーバライドしています．

一方のARM側では，DSPの16Mバイトのメモリ空間と同様のマッピングを論理空間に行うため，DSP空間のためのコピー領域を定義しています．この領域にはDARAMおよびSARAMがマッピングされているほか，DSP空間にSDRAMブロックをマッピングした場合と同じメモリ・ブロックをこのコピー領域にもマッピングします〔図5の（2）〕．プログラムのロードや通信データの読み書きなどのためにARM側からDSP空間にアクセスするときには，実際にはこのコピー領域にアクセスします．この領域では，DSP空間とオフセットが同じになるようにマッピングされているので，DSPとARM間でのアドレス変換は容易です．また，この領域はDSPとの共有メモリなので，キャッシュを無効にしておきます．

4. Linux上でのプログラミング

● 開発環境とLinux側のAPI

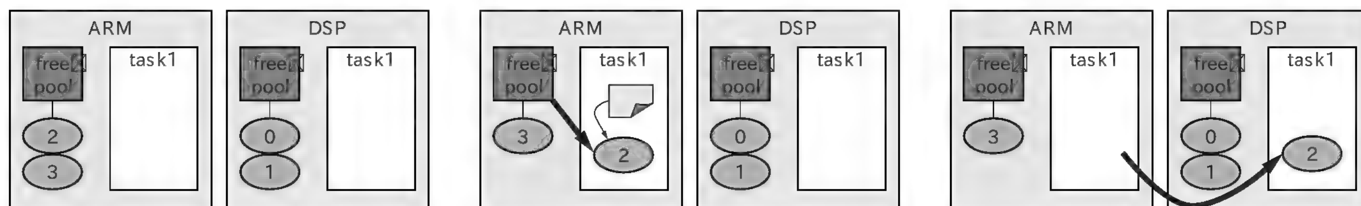
ARMサイドのプログラミングには，i386PCプラットホーム上で動くARM Linuxのクロス開発環境を使用します．

DSP GatewayのLinux側のAPIとして次の五つのデバイス・インターフェースが用意されています．

- DSP taskデバイス・インターフェース
  アプリケーションがDSPタスクと通信するためにアクセスするデバイス・インターフェース
- DSP task watchデバイス・インターフェース
  DSPタスクの状態を監視するためにDSPダイナミック・ローダが使用する[注1]
- DSP controlデバイス・インターフェース
  ユーティリティ・ツールがアクセスし，DSPのコントロールやコンフィグレーションを行う
- DSP watchdogデバイス・インターフェース
  DSPのウォッチ・ドッグ・タイマからの割り込みを検知する
- DSP exmemデバイス・インターフェース
  DSP空間にマッピングしたSDRAM領域にアクセスする．`dspctl`内のDSPプログラム・ローダが使用する

ここでは，このうちDSP taskデバイス・インターフェースとDSP contorolデバイス・インターフェースについて説明します．

（1）初期状態．ARM, DSPそれぞれ2ラインずつをfree poolに保持している
（2）ARM側のtask1がline2をfree poolから予約し，データを書き込む
（3）line2を使ってtask1がARMからDSPにデータを送信する



（4）DSPはARMからラインを受け取ると，代わりのライン（line0）のオーナ権をARMに譲りバランスする．DSP task1はline2からデータを読み出す
（5）DSP task1はデータを読み終えるとline2を解放し，free poolに返却する

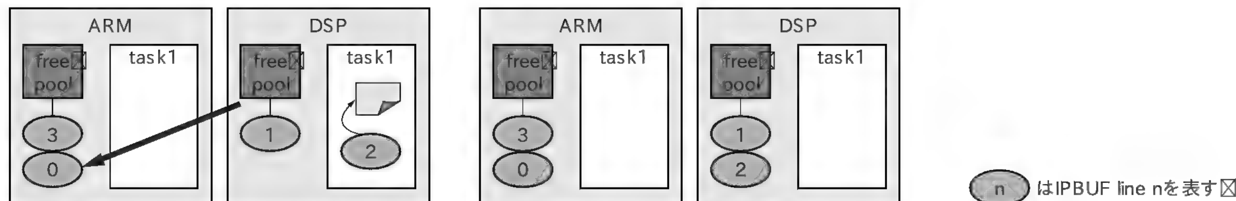


図4　Global IPBUF動作例
それぞれのプロセッサがバッファ数を自律的にバランスしながら通信する．そのバッファに対してオーナ権のあるプロセッサのみがアクセスすることができる．

● DSP taskデバイス・インターフェース

DSPを利用する通常のアプリケーション・プログラムがアクセスするデバイス・インターフェースです．DSP側のプログラムによって複数のDSPのタスク・デバイス・インターフェース(デバイス・ファイル)が作成されます．

たとえば，DSP側にMPEG-4のデコーダとMP3のデコーダの機能を実装したとすると，Linux側には/dev/dsptask/mpeg4，/dev/dsptask/mp3などのデバイス・ファイルが作成されることになります．これらのデバイス・ファイルは，DSP controlデバイス・インターフェースを通じてDSPシステムのコンフィグレーションを行うことで自動的に作成されます．

アプリケーション・プログラムは，これらのファイルに対して読み書きすることによりエンコード・データをDSPに送信したり，デコードされたデータをDSPから受け取ることができます．また，ioctlコマンドはMailboxコマンドを通じてDSPタスク内のTCTLコマンドとして通知されるので，ユーザがDSPタスク内に定義したTCTLコマンドをLinuxのioctlコマンドとして使用することができます．たとえば，タスクの動作モード変更を行うなど，フレキシブルなプログラミングが可能です(TCTLに関しては後述)．さらに，pollも実装されており，selectなどでDSPタスクからのデータを待つことができます．

このようにDSP Gatewayによって，Linux側のアプリケーション・インターフェースはシンプルでなじみやすいものになります．

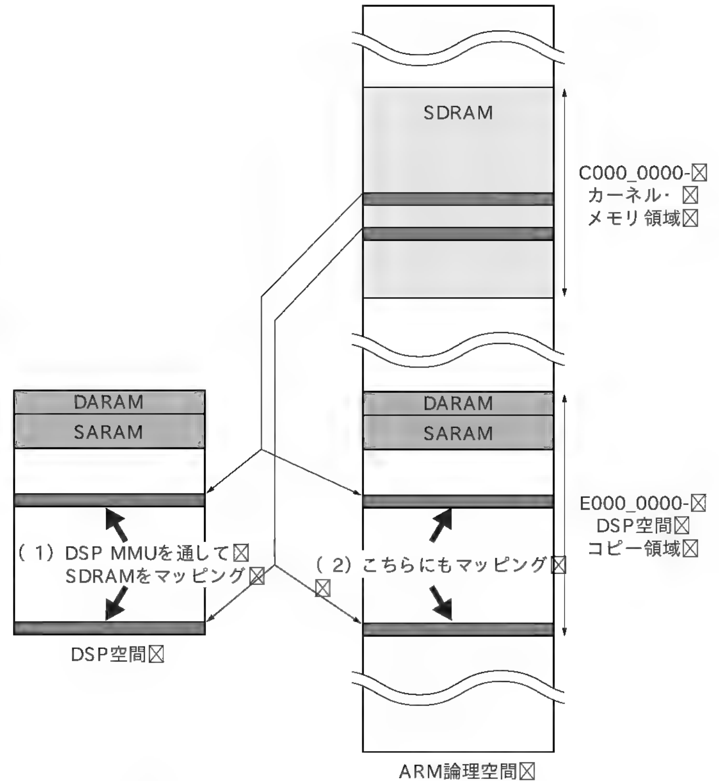


図5 ARM/DSPのアドレス空間
ARMの論理空間にDSP空間のコピーを作成し，共有メモリのアドレス変換を容易にしている．

● DSP controlデバイス・インターフェース

このデバイスは，おもにDSP Gateway付属のユーティリティであるdspctlがアクセスします．このデバイスを通して，ARM側からDSPのリセットやリセット解除，DSPメモリ空間へのSDRAMのマッピング，DSPシステム・コンフィグレーションなどを行います．

ファイル操作関数としては，open，releaseのほかにはioctlのみが実装されており，read()やwrite()は使用されません．

このデバイスで定義されているおもなioctlコマンドには以下のようなものがありますが，デバッグに使用されるコマンドもいくつか含まれます．

- **RESET，UNRESET**
  DSPのリセット，リセット解除を行う
- **SETRSTVECT**
  DSPのリセット・ベクタを設定する
- **IDLE**
  DSPをアイドル状態にする．設定によっては一部の回路をスリープ状態にする
- **DSPCFG，DSPUNCFG**
  DSPCFGはDSPアプリケーションおよびタスクのコンフィグレーションを行う．結果として/dev/dsptaskディレクトリにDSPのタスク・デバイス・ファイルが作成される．DSPUNCFGは作成されたDSPのタスク・デバイス・ファイルを削除し，ドライバ内のDSPタスク用リソースの解放などを行う
- **REGMEMR，REGMEMW，REGIOR，REGIOW**
  DSPメモリ空間，I/O空間のデータにアクセスする．デバッグ用
- **MMUINIT**
  DSP MMUの初期設定を行う．通常は自動で行われる
- **EXMAP，EXUNMAP**
  EXMAPはLinuxのページング・システムからメモリを確保し，DSP空間にマッピングする．EXUNMAPはマッピングを解除し，メモリを解放する
- **EXMAP_FLUSH**
  EXMAPなどによって行われたDSP空間へのマッピングをすべて解除し，初期状態にする
- **FBEXPORT**
  Linux側で定義されたフレーム・バッファをDSP空間にマッピングし，DSPタスクが直接フレーム・バッファにアクセスできるようにする
- **MBSEND**
  DSPに対して，Mailboxコマンドを手動で発行する．デバッ

注1：DSPダイナミック・ローダは現在開発中．本稿では解説しない．

グ用

● /proc デバイス・インターフェース

/proc/dsp 以下にいくつかの proc デバイス・ファイルが作成されます(その多くは DSP プログラムのコンフィグレーション後に作成される)．おもにデバッグ情報を得るためのもので，DSP Gateway そのものや，DSP アプリケーションが正しく動作しているかどうかをチェックすることができます．以下にその一部を紹介します．

● **/proc/dsp/mblog**

Mailbox コマンドのログ．どのようなコマンドが ARM と DSP の間でやりとりされたのかを見ることができる

● **/proc/dsp/icrmask**

DSP がアイドル状態になるときに，ICR レジスタに書き込む値を設定する．この値は DSP サブ・システムのどのブロックをスリープさせるかを決定する

● **/proc/dsp/mmu**

DSP MMU の状態を見ることができる

● **/proc/dsp/ipbuf**

Global IPBUF の状態を見ることができる

● **/proc/dsp/task/<taskname>/status**

DSP タスクの状態を見ることができる

● **/proc/dsp/task/<taskname>/fifosz**

DSP タスクの読み込み用 FIFO のサイズを見たり，変更したりすることができる

5. DSP プログラミング

● 開発環境――CCS の DSP/BIOS を使う

TMS320 系の DSP のアプリケーション開発ツールとして，TI 社は Code Composer Studio(CCS)という統合開発環境を提供しています．この中には，C/C++ およびアセンブリのコンパイラ，基本ライブラリ，そして DSP/BIOS と呼ぶリアルタイム・ソフトウェアを構築するためのライブラリなどが含まれています．DSP Gateway もこの DSP/BIOS を利用しており，DSP タスクは DSP/BIOS の TSK スレッドとして動作し，マルチタスク環境が実現されています．DSP/BIOS については参考文献(3)を参照してください．

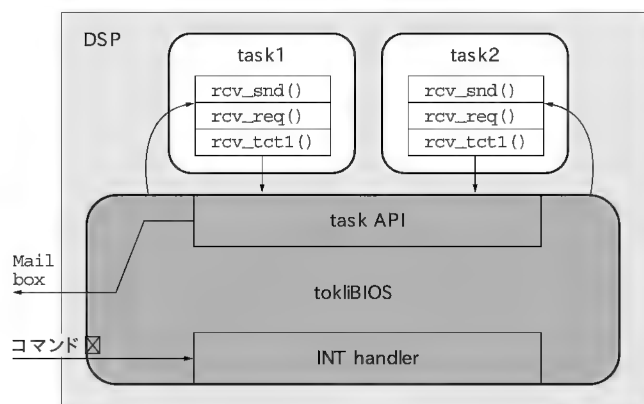


図6　DSP アプリケーション構成
tokliBIOS の上に複数のタスクを実装することができる．これらはマルチタスクで動作する．

● **tokliBIOS** ライブラリ

DSP Gateway では，DSP 側のアプリケーションは図6のような構成になっています．コアに tokliBIOS と呼ばれるライブラリがあり，ARM との通信や IPBUF の管理，割り込み処理，タイマ管理，パワー・マネージメントなどを行います．また，ARM からの Mailbox コマンドに応じて DSP タスクに実装された関数を呼び出します．

一方，DSP タスクに対して API を提供し，タスクはそれを通じて ARM とデータを通信したり，IPBUF の予約や解放などを行います．DSP タスクとは，DSP で実現する機能の処理単位となるもので，tokliBIOS の上で動作するアプリケーションとしてユーザが実装します．

以下では DSP タスクの実装方法について説明します．

● DSP タスク・プログラミング

DSP タスクは下記の dsptask 構造体で定義されます．

```
struct dsptask {
        Uns     tid;
        String  name;
        Uns     ttyp;
        Uns     (*rcv_snd)();
        Uns     (*rcv_req)();
        Uns     (*rcv_tctl)();
        struct TSK_Attrs *tsk_attrs;
}
```

ここで Uns や String のデータ型は CCS 付属のヘッダで，それぞれ unsigned char* として宣言されています．同様に Int は int，Void は void，LgUns は unsigned long などになります．構造体のメンバの説明の前に，タスクのタイプについて説明しておきます．タスク・タイプは，DSP タスクにて実現する機能の性質によってプログラマが決定します．

● DSP のタスク・タイプ

DSP タスクのタイプは，以下の項目の組み合わせによって定義されます．

● **Active Sending/Passive Sending**

Active Sending タスクは，能動的にデータを ARM 側に送信する．逆に Passive Sending タスクは，ARM 側から要求されないかぎりデータ送信を行わない．Linux 側で read が呼ばれると，それをきっかけに本タスクに対してデータ・リクエストが発行され，タスクが ARM に対してデータを送信する．典型的な CODEC などのタスクは，Active Sending タスクになることが多くなる

● **Active Receiving/Passive Receiving**

Active Receivingタスクは，タスク内で明示的にデータを要求しないかぎりARM側からデータを受け取らない．Linux側の動作としては，タスクがデータ要求を発行するまでは本タスクに対するwriteコールがブロックすることになる．逆にPassive Receivingタスクに対しては，ARMからのwriteはブロックすることなく，いつでもデータを送信することができる．典型的なCODECなどのタスクは，Passive Receivingタスクになることが多くなる

●**Word Sending/Block Sending**

Word Sendingタスクはワード単位でデータを送信する．送信時にIPBUFは使用しない．Block Sendingタスクは IPBUF(Globalまたは Private)を使用してブロック・データを送信する

●**Word Receiving/Block Receiving**

Word Receivingタスクはワード単位でデータを受信する．受信時にIPBUFは使用しない．Block Receivingタスクは IPBUF(Globalまたは Private)を使用してブロック・データを受信する

●**Global Block Sending/Private Block Sending**

Global Block Sendingタスクは Global IPBUFを使用してデータを送信する．Private Block Sendingタスクは Private IPBUFを使用してデータを送信する

●**Global Block Receiving/Private Block Receiving**

Global Block Receivingタスクは Global IPBUFを使用してデータを受信する．Private Block Sendingタスクは Private IPBUFを使用してデータを受信する

● dsptask構造体のメンバ

それでは，以下にdsptask構造体のメンバについて説明します．

●**Uns tid;**

タスクID．初期値にはTID_MAGICを設定する

●**String name;**

タスクの名前．ここに指定された名前がLinux側のデバイス・ファイル名になる

●**Uns ttyp;**

タスクのタイプを設定する

タスクの動作を決める関数．原則的にはそれぞれ，Linuxアプリケーションが対応するタスク・デバイスにwrite，read，ioctlを発行したときに呼ばれるが，タスクのタイプによっては使用されない関数もある．

●**Uns (*rcv_snd)();**

rcv_snd()は，Word Receivingタスクでは受信ワードを，Block Receivingタスクでは受信IPBUFラインのBIDが引き数として呼ばれ，タスクのデータ受信時の動作を定義する．Active Sendingタスクでは本関数内でbksnd()，またはwdsnd()というAPI関数を使ってARMにデータを送信する

●**Uns (*rcv_req)();**

rcv_req()は，Passive Sendingタスクでのみ呼ばれる．Linux側でreadが呼ばれるとMailboxコマンドを通じてこの関数が呼ばれるので，wdsnd()やbksnd()などのAPI関数を使ってARMにデータを送信する．Active Sendingタスクでは本関数は呼ばれないので，NULLを設定する

●**Uns (*rcv_tctl)();**

rcv_tctl()は，Linux側でioctlが呼ばれるとMailboxのTCTLコマンドとなってこの関数が呼ばれる．引き数としてioctlで指定したコマンドが渡されるので，Linux側からタスクに対してioctl的なコントロールをしたい場合にユーザがそのコマンドおよび動作を定義することができる

●**struct TSK_Attrs *tsk_attrs;**

タスクのDSP/BIOSで定義される属性．たとえばタスクの優先順位などを決める

● DSPのタスクAPI

tokliBIOSライブラリには，DSPタスクのAPIとして以下のようなものが用意されています．IPBUFの予約や開放，ARMとのデータ授受などが含まれます．

●**Uns *ibpuf_d[];**

IPBUFデータへのポインタ．添え字にはBIDを指定する．get_free_ipbufで確保した，もしくはGlobal Block Receivingタスクが BKSNDで受け取ったIPBUFラインのデータにアクセスすることができる

●**Uns get_free_ipbuf();**

IPBUFを予約する

●**Void unuse_ipbuf();**

IPBUFを開放する

●**Void wdsnd();**

Word Sendingタスクが1ワードのデータをARMに送信する

●**Void bksnd();**

Global Block Sendingタスクが IPBUF内のデータをARMに送信する

●**Void bksndp();**

Private Block Sendingタスクが IPBUF内のデータをARMに送信する

●**Void bkreq();**

Global Block Receivingタスクが ARMにデータを要求する

●**Void bkreqp();**

Private Block Receivingタスクが ARMにデータを要求する

●**Void cmderr();**

ARMにエラーを通知する

●**Void dbg();**

ARMにデバッグメッセージを送信する

● DSPタスクの動作

DSPタスクの動作をもう少し詳しく見ていきます．

図7のように，tokliBIOS内部では各タスクごとにDSP/BIOSのTSKスレッドが生成され，これらがDSPタスクの関数を実行する実体となります．各スレッドはMBQ(mailbox queue)というキューを内部にもっており，ARMからのMailboxコマンドは各スレッドのMBQに振り分けられます．

MBQにコマンドが登録されると，スレッドはそのコマンドに応じたDSPタスクの関数を呼び出します．ARM側から一つのタスクに対してMailboxコマンドを連続して(たとえばBKSND，BKREQ)送信したとしても，それらはMBQにたまり，スレッドはコマンドを順番にひとつひとつ処理します．したがって，たとえば送られてきたデータを加工して返すタスクに対してBKSNDコマンドとBKREQコマンドが連続して発行されたとしても，後のBKREQではきちんと処理されたデータを返すことができるのです．

MBQが空のときは，そのスレッドはスリープ状態になり，スケジュール対象にはなりません．

6. DSP Gatewayを使った動作例

DSP Gatewayを使用することにより，DSP上で動作するさまざまなCODECやフィルタをARM側のLinuxから利用することができます．

ここでは，実際にサンプル・プログラムを作成して，Linux上のアプリケーションからDSP側で動作するプログラムを利用する例を説明します．

ここで紹介する例では，DSP側にローパス・フィルタとして実装されたFIRフィルタをLinuxのアプリケーションから利用します．

● CCSプロジェクト

DSP側のプログラムを作成するために，TI社のCCSを使用して新規プロジェクトを作成します．プロジェクトのBuild optionsにおいて，表2に示すオプションを設定します．

このプロジェクトには，表3に示すファイルを登録します．

実際のソース・コードは，Webにアップロードします．各ファイルがプロジェクトへ登録されたようすを図8に示します．

● DSPタスクの作成

Linux側アプリケーションのFIRフィルタに対するインターフェースとなるDSPタスクを作成します．dsptask構造体の変数を定義することにより，DSPタスクを作成します．ここでは，dsptask構造体の変数を次のように定義しました．

```
struct dsptask task_fir = {
  TID_MAGIC,        // Task ID
  (String)"fir",    // Task Name
  MBCMD_TTYP_BKDM | MBCMD_TTYP_BKMD,
                    // Task Type
  fir_bksnd,        // Function for write
  fir_bkreq,        // Function for read
  fir_tctl,         // Function for ioctl
  &fir_attr         // Task Attribute
};
```

タスクIDには，tokliBIOS.hヘッダ・ファイルで定義されている“TID_MAGIC”を指定します．このタスクは，FIRフィ

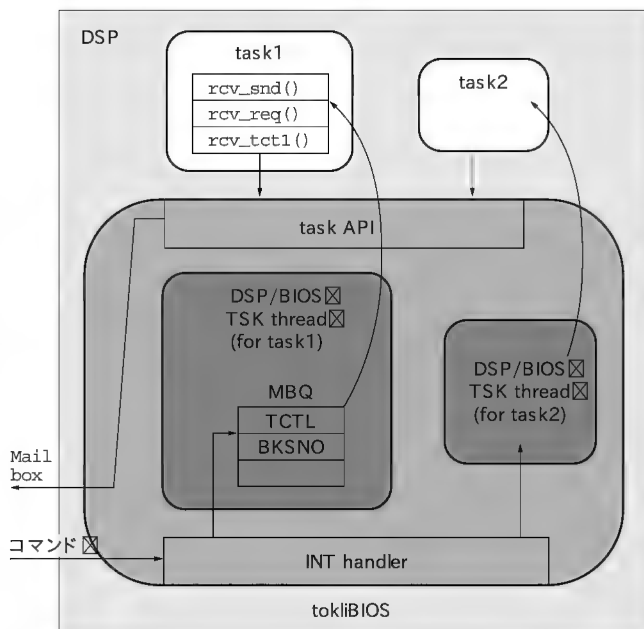


図7 DSPタスクとMBQ
タスクの動作実体であるTSKスレッドは，キューに登録されたコマンドに従ってタスクに実装された関数を呼び出していく．

表3 各ファイルの内容

ファイル	内 容
fir.c	DSPタスクの定義，Linux側のシステム・コールに対応する関数の定義，FIRフィルタへのインターフェースの実装
firasm.asm	FIRフィルタ本体
sysinfo.c	定義したDSPタスクのDSP Gatewayへの登録
fir.cmd	IPBUFやそのほかのプログラム内で使用するシンボルのメモリへの配置を指定

表2 CCSプロジェクトのBuild options

タブ	カテゴリ	アイテム	アクション
Compiler	Advanced	Large memory model	選択
	Preprocessor	Include Search Path	dsplib.hとtokliBIOS.hのあるディレクトリを追加
Linker	Basic	Library Search Path	55xdspx.libとtokliBIOS.libのあるディレクトリを追加
		Include Libraries	55xdspx.libとtokliBIOS.libを指定

ルタを実現するものなので，名前を“fir”とします．タスクのタイプは先述したように，Linux 側のアプリケーションとの通信の方式により決定します．

今回使用する FIR フィルタでは，DSP タスクへのデータ送信は Linux 側のアプリケーションが主導的に動作して行われます．また，Linux 側のアプリケーションへのフィルタ処理結果の送信は，DSP タスクがフィルタ処理終了時に能動的に行います．このデータの送受信には Global IPBUF を使用します．

また，この FIR フィルタは入力データと出力データを float 型の配列として受け渡します．そのため，このタスクのタイプは，Active Block Send かつ Passive Block Receive と指定します．Linux 側のシステム・コールに対応する関数は，必要に応じて指定します．今回は，write/ioctl システム・コールのそれぞれに対する関数を指定しました．タスクの属性はデフォルト値を使用してください．

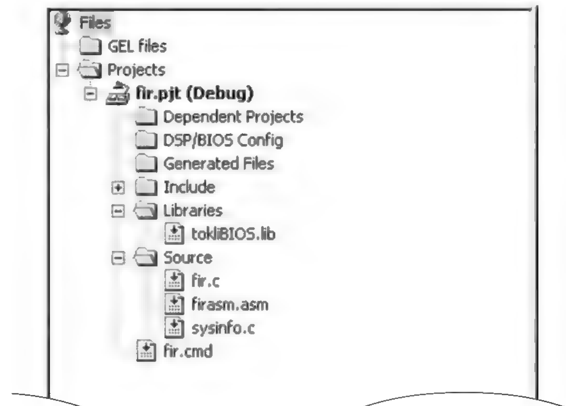


図8　プロジェクトに登録されたファイル

● DSP 側のプログラミング

DSP Gateway で提供するインターフェースは，read/write/ioctl とシンプルなので，Linux 側から DSP 側のプログラムを複雑に制御する必要のある場合には，ioctl をうまく利用することが重要になります．今回使用する FIR フィルタは次のような処理を行います．

1) 初期化
2) データ数の受信
3) フィルタのタップ数の受信
4) データ処理用バッファの割り当て
5) フィルタ係数の受信
6) データの受信
7) フィルタ処理の実行
8) 処理結果の送信

ここで注目すべき点は，この DSP タスクは 4種類のデータを受信する必要があることです．DSP タスクには，Linux 側の write システム・コールにより呼び出される関数は一つしかありません．そこで，DSP 側に DSP タスクの状態を示す変数を持たせ，事前に Linux 側から ioctl システム・コールを使って DSP タスクを適切な状態へ遷移させます．その後，該当するデータを write システム・コールにより送信します．

DSP 側では，この状態変数を参照することにより，受信したデータの種類がわかるようになります．この例では，Linux 側のアプリケーションからコントロールを可能にするために以下の ID を定義しました．

```
#define FIR_INIT          0x0080
#define FIR_NUM_DATA      0x0081
#define FIR_NUM_TAPS      0x0082
#define FIR_WRITE_DATA    0x0083
#define FIR_WRITE_COEF    0x0084
#define FIR_MEM_ALLOC     0x0085
```

それでは，具体的に Linux 側のアプリケーションが read/write/ioctl システム・コールを呼び出したときに実行される関数の実装について説明します．

read システム・コールに対応する関数には，今回作成した DSP タスクのタイプが Active Sending タイプなので，NULL ポインタを指定します．

write システム・コールに対応する関数 fir_bksnd では，状態変数が示す内容に応じて，データ数，タップ数，フィルタ係数，データの受信を識別し，該当するバッファへ各データを格納します．また，状態変数がデータの受信を示すときは，データの受信後にフィルタ処理の実行をして，その処理結果を Linux 側へ送信します．

ioctl システム・コールに対応する関数 fir_tctl では，パラメータとして渡された値で状態変数を更新します．さらにその状態に応じて，初期化，バッファの割り当てを行います．

● Linux 側のアプリケーション

このアプリケーションでは，正弦波に高周波のノイズが乗ったデータを疑似的に生成し，FIR フィルタへの入力データとします．それを DSP Gateway を介して DSP 側のプログラムへ転送し，FIR フィルタ処理を実行させ，その結果をファイルへ格納します．このソース・コード(fir_app.c)は，本誌の Web サイトにアップ・ロードします．

このアプリケーションは，DSP 側のプログラムに対して以下の処理を行います．

open　: /dev/dsptask/fir ファイルのオープン
ioctl : 初期化
ioctl : データ数送信のための状態変更
write : データ数の送信
ioctl : フィルタのタップ数送信のための状態変更
write : フィルタのタップ数の送信
ioctl : データ処理用バッファの割り当て
ioctl : フィルタ係数送信のための状態変更
write : フィルタ係数の送信

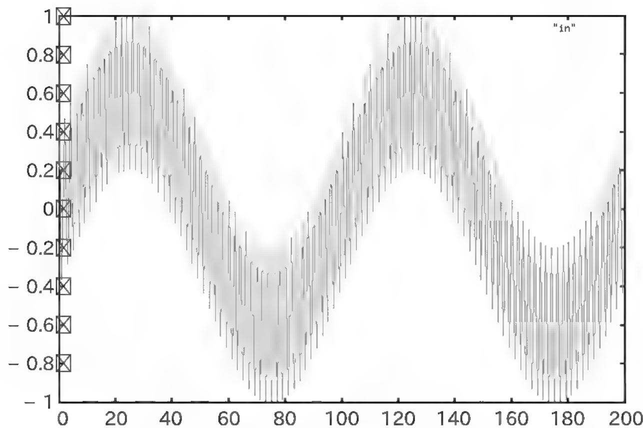


図9　フィルタへの入力データ
正弦波に高周波のノイズが乗っている．

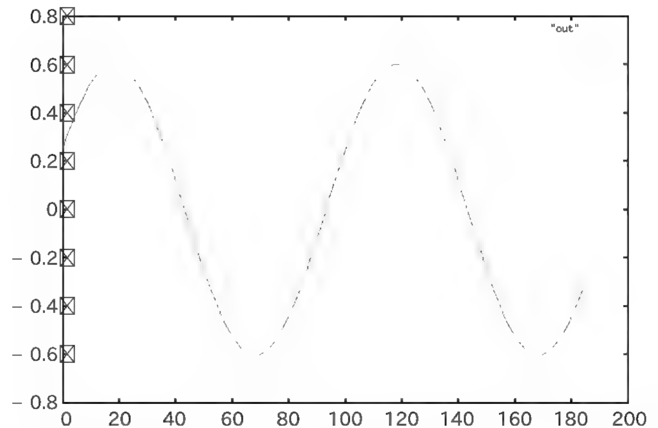


図10　フィルタの処理結果
ノイズが取り除かれ，きれいな正弦波が出力されている．

`ioctl`：データの送信のための状態変更
`write`：データの送信
`read` ：処理結果の受信
`close`：`/dev/dsptask/fir`ファイルのクローズ

● **実行例**

今回作成したプログラムは次の二つです．

- Linux側のアプリケーション：`fir`
- DSP側のプログラム　　　：`fir.out`

Linux上でDSP Gateway付属のユーティリティである`dspctl`を使用して作成したDSP側のプログラムを実行します，

```
# dspctl load fir.out
# dspctl unreset
# dspctl dspcfg
# fir
```

`fir`アプリケーションが正しく実行されると，カレント・ディレクトリに“`in`”ファイルと“`out`”ファイルが作成されます．“`in`”ファイルには，図9に示すFIRフィルタへの入力データが格納されます．“`out`”ファイルには，図10に示すFIRフィルタの処理結果が格納されます．

おわりに

Linuxが組み込みシステムのOSとして検討されるようになって数年になります．組み込みシステムはPCとは違った専用のハードウェアを持つことが多く，オープン・ソース・コミュニティの広がりという意味ではユーザ，開発者ともに限られたものになりがちです．それでもLinuxは確実にその守備範囲を広げており，今では代表的なCPUに関してはほとんどすべてがサポートされているといってよいでしょう．

DSPはCPUではありませんが，組み込みシステムを構成する重要なコンポーネントです．LinuxがDSPのようなコンポーネントを多くサポートするようになることで，組み込みLinuxの発展につながるものと期待しています．

誌面のつごうで書ききれませんでしたが，DSP Gatewayはまだまだ多くの機能をもっています．代表的なものとして，DSPからのダイレクトIOのサポート，DSPのパワー・マネージメント，フォルト・トレラント・サポート，zero copy/mmapインターフェースのサポート，DSPタスクのダイナミック・ローディングのサポートなどがあります．

DSP Gatewayは開発が進められている最中なので，新しい機能は随時追加され，公開されていく予定です．

興味のある方は，ぜひメーリング・リストに参加して議論に加わってください．お待ちしています．

参考文献
(1) Toshihiro Kobayashi；Linux DSP Gateway Specification Rev 2.0, Nokia
(2) spru683:OMAP5910 Dual-Core Processor Inter-Processor Communication Reference Guide, Texas Instruments
(3) spru423:TMS320 DSP/BIOS User's Guide, Texas Instruments

もり・えいご/こばやし・としひろ/たかはし・きよたか
ノキア・ジャパン(株) ノキア・リサーチセンター

米国マクデータ社ジョン・ケリー氏に聞く

iSCSI技術のストレージ製品への応用

北村 俊之

ストレージ業界では，以前から「iSCSI（Internet Small Computer System Interface：アイスカジー）」が話題になっている．このiSCSIは，2003年2月にインターネット技術の標準化団体「Internet Engineering Task Force（IETF）」によって正式な規格として承認された．現在のストレージ・エリア・ネットワーク（SAN）環境では，どちらかというとクローズなアーキテクチャといわれているファイバ・チャネル（Fibre Channel：FC）によるデータ伝送が主流となっている．これに対して，オープンなアーキテクチャを軸に，既存のIP技術を応用して安価なシステム構築を可能にするiSCSI技術に，業界全体の期待が高まっているといえそうである．

では，このiSCSIの規格化によって，現在のSAN環境はどのような変化をとげるのだろうか．今回は，iSCSIの動向を探るとともに，米国マクデータ社取締役会長であるジョン・ケリー氏に同社のSAN戦略を聞いた．

ストレージ・システムとは ―― NAS（IP）～SAN，SAN～iSCSI（IP）へ

これまでのストレージ・システムには，大きく分けて三つの種類が存在していた．サーバに直接接続する「DAS（Direct Attached Strage）」，サーバと同一ネットワーク上に接続する「NAS（Network Attached Storage）」，サーバとは別のストレージ用ネットワークを構築する「SAN（Storage Area Network）」がそれである．NASのハードウェア構成は，PCやワークステーションと同等である．Windowsなど既存のOSの機能を利用したファイル共有と異なる点は，組み込み向けOSを搭載することで，設定の簡素化や安定性を実現したことだろう．この方式によって，従来のファイル共有より安定性のあるストレージ・システムの構築が実現できるようになったが，その反面，ネットワーク上のトラフィックが増大するという問題が出てきた．

この問題を解決するために，ストレージ系ネットワークを通常のIPネットワークから切り離してしまうSANという方式が考案され，現在ストレージ・システムの主流となっている．SANでは伝送路としてEthernetではなく，光ファイバを利用して800Mbpsの伝送速度を実現する「FC-AL（Fibre Channel Arbitrated Loop）」が用いられている．このFC-ALでは，FCスイッチを利用することで，数kmまでの遠隔接続が可能となるなどのメリットがうたわれる反面，既存のIPネットワークとは独立したFC-ALネットワークを構築しなければならないため，機器導入のコストが高いといったデメリットが指摘されている．

このように，ここ数年FC-SANが主流となっていたが，最近Ethernetを利用したIP-SANが，再び脚光を浴びはじめている．その理由の一つとしては，1Gbps～10Gbpsの伝送速度を実現する「ギガビットEthernet」の実用化があげられる．従来の10倍～100倍の伝送能力をもつギガビットEthernetの使用を前提として，再びIPネットワークにストレージ系ネットワークを統合しようとする動きである．そのもっとも代表的な技術が「iSCSI」というわけだ．

もちろん，IPネットワークをベースとしたSANはiSCSIだけではない．既存のFCをIPネットワークと融合させるための「Fibre Channel over IP（FCIP）」や「Internet-Fibre Channel Protocol（iFCP）」などがすでに実用化されている（表1）．

ルーセント・テクノロジーを主体とした数社で共同開発されたFCIPは，ファイバ・チャネル制御コードとデータをIPパケットに変換し，遠隔地のFC-SAN間でのデータ転送を可能にするものである．ここではIPとFCは，ギガビットEthernetポートを搭載したFCスイッチで相互接続される．遠く離れた場所にあるFC-SANどうしを接続することで，LANとSANを一つのIPネットワークに包括しようというわけである．

また，ニシャンシステムズが提唱したiFCPでは，FCアドレスとIPアドレスをマッピングしてルーティングを行うというものである．これによって，Any-to-AnyのIPルーティングができるようになっている．なお，ニシャンシステムズは，2003年にマクデータに買収され，そのIP技術は同社に継承されている．

iSCSIとは

iSCSIをひと言で表現するならば，「IPパケットで包んだSCSIコマンドを介して，コンピュータとストレージ間のデータ通信

表1
ファイバ・チャネルとIP-SANの比較(マクデータ社提供)

	ファイバ・チャネル	IP-SAN
ネットワーク	ファイバ・チャネル(FC)	IPベース・ネットワーク
プロトコル	FCP	iSCSI/iFCP/FCIP
HBA注(Initiator/Target)	FC専用のHBA(各ベンダ)	NIC/iSCSI専用のHBA(各ベンダ)
デバイス・ドライバ	FC専用のドライバ(各ベンダ)	iSCSI専用のドライバ(各ベンダ)
データ転送速度	1Gbpsまたは2Gbps	100Mbps, 1Gbps, 10Gbps
ファイル・システム	サーバに依存	サーバに依存
CPUのオーバヘッド	小	高(TOEチップ搭載のHBAにより，小)
I/Oの方式	ブロック単位	ブロック単位(ファイル・ベース転送方式を活用)
I/Oのトラフィック	小	大
接続距離	FCダイレクト接続で10km	無制限
長距離(WAN)対応	専用の装置が必要(WDMなど)	可能(iFCP，FCIPを利用)
適用範囲	高パフォーマンスの要求 I/Oトラフィックの高い環境	低・中パフォーマンス環境 I/Oトラフィックの要求を重視しない
コスト	やや高	やや低

注：HBA；Host Bus Adapter

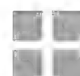
をIP上で行うためのプロトコル規格」ということになるだろうか．インターネット経由で複数のデータ・ストレージ・システムをネットワークに接続できることで，より高速なSANを実現する基盤として注目を集めている．

これまでのSCSIでは，各種制御やデータ転送のためのコマンドとともに物理的な伝送路も規定されていた．これに対して，iSCSIではSCSIのコマンド部分を取り出し，伝送路にIPネットワークを採用することで汎用性を高めている．たとえば，FCIPはFC技術間でしか利用ができないが，既存のEthernetとの間に相互運用性があるiSCSIは，安価なEthernet用スイッチング・ハブやルータなどを利用することで，サーバ-ストレージ間の接続を簡単に，しかも低コストで実現できる．本社-バックアップ・センタ間など遠隔地からのデータ共有のリアルタイム処理，災害時におけるシステムの復旧対策(ディザスタ・リカバリ)などに使えるとされている．現在，SANの伝送路として主流となっているFCに変わる存在として，もっとも期待が高まっている技術であるとされているのは，こうした理由によるところが大きい．

また，既存のネットワーク技術の管理手法がそのまま利用できるため，導入や運用，管理面でのコストが大幅に削減できるという点も大きなメリットとされている．iSCSI対応のストレージ・デバイスは，IPネットワークに直接接続されることになる．ここにアクセスするPCやワークステーションに必要なものは，ネットワーク・インターフェースとiSCSIに対応したデバイス・ドライバだけである．別途ストレージ用のインターフェースを用意する必要はない．また，ケーブル長の制限もない．すでにギガビットEthernetが導入されている環境であれば，iSCSIに対応したストレージ・デバイスを直接接続し，クライアントにデバイス・ドライバを組み込むだけである．ストレージ系専用のIPネットワークを別途に構築する場合でも，ギガビットEthernet製品がそのまま利用できるため，FC-ALの新規導入と比較して大幅なコストの削減が期待できる．

またエンタープライズ用のデータベースでは，伝送速度を上げるために，ディスク・アクセス時にファイル・システムを経由せず，ディスクに直接アクセスする「RAWアクセス」方式をとっている製品もある．iSCSIは見かけ上，従来のSCSIと変わらないため，NASでは不可能だった，こうした特殊なディスク・アクセス方式にも対応できる．さらに，伝送速度が品質に影響を及ぼすストリーミング映像配信などへの応用にも期待が高まっている．こうしたiSCSIの特徴は，これまでFCを利用して構築されてきたストレージ・システムを，より低コストで構築できるという点である．

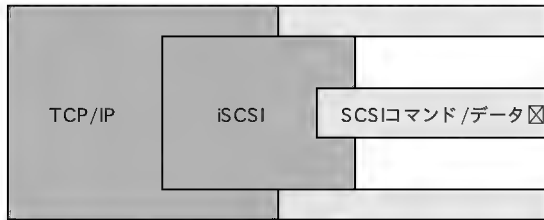
米国の調査会社であるGartner Dataquestでは，SANとサーバの接続は，2006年までにiSCSIによるものがほかの競合技術に勝り，導入規模がサーバ数で約150万台に達すると予測している．また，IDCでは，iSCSIアレイ市場について，2003年の2億1600万ドル規模から，2007年には50億ドル規模に急成長すると予測している．



iSCSIのしくみ

それでは，ここでiSCSIのしくみを簡単に見てみよう．iSCSIは，ストレージとネットワークで使用されている二つのプロトコルから構成される．ストレージ側ではSCSIコマンド・セットが使用され，ネットワーク側ではIPとEthernetが使用される．これらは，現在，企業ネットワークなどで基礎をなす技術であり，MAN(Metropolitan Area Network)やWAN(Wide Area Netowork)の構築でも幅広く利用されている．iSCSIレイヤを含むプロトコル・スタックの構造(図1)を簡単にいえば，まず，従来のSCSIコマンドをトランスポート層のTCPでカプセル化し，TCP/IPヘッダを付加した後，IPパケットとして扱えるようにするということになる．

iSCSI-SANを構築するための要素として，まずファイル・サーバなどの「iSCSIイニシエータ」および，ディスク・アレイや磁気テープ・サブシステムなどの「iSCSIターゲット」が挙げられる．iSCSIイニシエータはSCSIコマンドを送出可能なSCSI



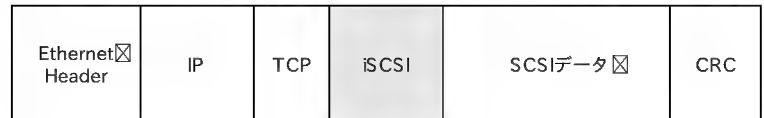


図1　iSCSIパケットの構造

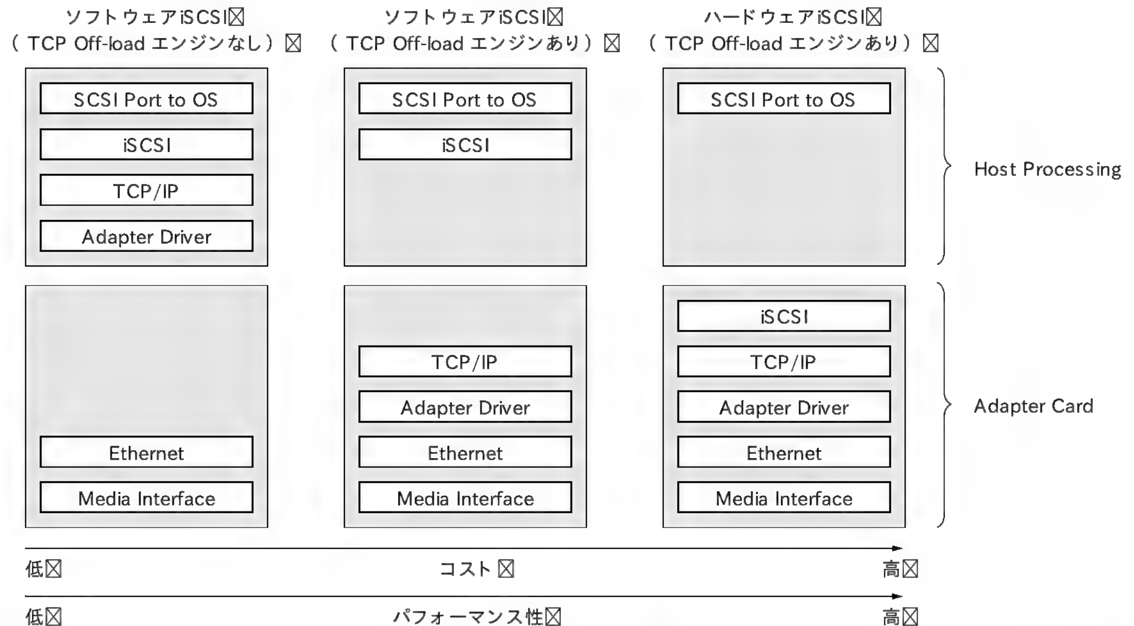


図2
iSCSIアダプタの構成
（性能と価格）

ホスト，iSCSIターゲットはSCSIコマンドを実行可能なSCSIデバイスということになる．こうした機器には，ホスト機器の処理負荷を低減するTCP/IPオフロード・エンジンなどのプロトコル処理機能が搭載されている場合もある（図2）．イニシエータ-ターゲット間の各セッションは，一つまたは複数のコネクション上で実行され，各コネクションではログイン・フェーズが実行される（図3）．このログイン・フェーズでは，iSCSIイニシエータ・ネームを利用して，ユーザの確認作業などを行うことができ，暗号化の指定，IPSecを利用したIPプロトコル・レベルでの暗号化なども行うことができる．

ファブリック[注1]も，重要な構成要素の一つとされている．IPをベースとしたファブリックの利点は，SANファブリック上のデータ搬送に標準的なEthernetスイッチとEthernetルータを使用できることである．また，このファブリックは，iSCSIインターフェース，およびそのほかのSCSIやFCなどのストレージ・インターフェースの組み合わせをサポートした，ストレージ・スイッチとストレージ・ルータを含んでいる．このストレージ・スイッチとストレージ・ルータによって，従来のIPとEthernetスイッチでは不可能だった，マルチプロトコルの接続性を実現している．

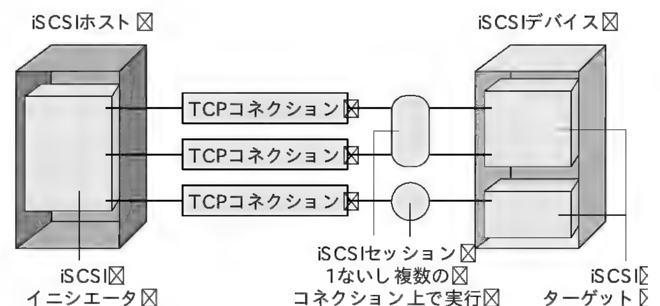
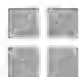


図3　iSCSIの論理的な接続形態
iSCSIプロトコルが，セッション中の配信順番を保ったり，セッションのリカバリを行う．

iSCSI-SAN構築のもう一つの要素として，SANの相互接続が挙げられる．iSCSIはネイティブなIPベースのプロトコルなので，SANの相互接続にはストレージ特有の機能を必要としない．共有型もしくは専用型のIPおよびEthernetネットワークをそのまま使用できる．

注1：FCで用いられるスイッチング装置．



iSCSIの実用化に向けて

IETFによるiSCSI規格の正式な認定を受けて，2003年中旬から，ネットワーク・ベンダ，ストレージ・ベンダなどによるiSCSI実用化に向けての実証実験が開始された．2003年6月には，NTTコミュニケーションズ，日本アイ・ビー・エム，シスコシステムズ，プロストレージの4社による，長距離・広帯域IPネットワークとiSCSIストレージを組み合わせた，遠隔地（東京〜名古屋間）におけるストレージ・ネットワークの検証実験が発表された．ここでは，NTTコミュニケーションの長距離・広帯域IPネットワーク上で，シスコシステムズのiSCSI対応ストレージ・ネットワーク・スイッチと日本アイ・ビー・エムのオープン・ストレージ装置を組み合わせて検証実験が行われた．ハードウェアやソフトウェア，ネットワークの接続性の検証や広域ストレージ・ネットワークの基本性能試験，ストレージ・ネットワークと仮想化技術を組み合わせた実験などがおもに行われた．

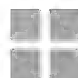
また，同年7月には，NEC，マイクロソフト，NTTコミュニケーションズの3社による，「Microsoft Windows Server 2003」上において，iSCSIを利用した広域ストレージ・ネットワークの共同実証実験が発表されている．この実証実験では，NTTコミュニケーションズの超広帯域IPネットワークとNECのIAサーバを用い，東京〜名古屋間の遠隔環境における接続確認や転送性能の測定が行われた．また，これらのネットワーク検証を実施したうえで，マイクロソフトの「Microsoft SQL Server 2000」や「Microsoft Exchange Server 2003」を用いて，ストレージ統合，ディザスタ・リカバリやレプリケーションなど，実運用環境を想定した実証検証も行われた．こうした実証実験により，これまで建物や敷地内など利用可能場所に制限のあったストレージ装置が，場所を問わずに遠隔地などから利用できることが実証されたことになる．

そして，11月には，NTTと日立製作所が，企業オフィスを災害などから守る「ストレージ・セントリック・ネットワーク」のプロトタイプの共同開発を発表している．ここでもクライアントPCとストレージの接続には，iSCSIが採用され，広域EthernetやIP-VANで広域接続が可能なソリューションを提示している．こちらは，2005年の商用化を目指しているという．

iSCSI技術に注目しているのは，ストレージやネットワークといったベンダだけではない．マイクロソフトは，2003年7月にWindows用のiSCSIソフトウェア・パッケージ「iSCSI Software Initiator version1.0」を公開している．同パッケージには，「IPsec」を含むデータ暗号化のサポートや，ストレージの特定に用いるサーバおよびクライアント用の「iSNS」，マイクロソフトの情報管理フレームワーク「WMI」による管理，さまざまなハードウェア・イニシエータを共通フレームワークに集約するためのアーキテクチャなどが含まれている．同パッケージは，Microsoft Download Centerで公開されており，無償でダウンロードできる．動作対象となっているOSは，「Windows 2000 Server/Workstation」および「Windows XP」，「Windows Server 2003」で，x86とIA-64アーキテクチャをサポートしている．

現在，多くの独立系ソフトウェア・ベンダ（ISV）や，独立系ハードウェア・ベンダ（IHV）が，Windows用のiSCSIアプリケーションおよびストレージ・ハードウェア製品の開発に着手しているといわれている．マイクロソフトは，こうしたIHVが開発するiSCSIハードウェア製品に対して，Windows製品と相互運用性を持ち，信頼性の高さを認定する「iSCSI Designed for Windows Logo Program」も開始しており，2003年11月の段階で，アダプテック，シスコシステムズ，インテル，マクデータ，ネットワークアプライアンスなど14社が認定を受けている．さらにマイクロソフトでは2003年10月に，ストレージ・システム向けOS「Microsoft Windows Storage Server 2003 日本語版」を発表している．アイオメガ，EMCジャパン，デルコンピュータ，NEC，日本ヒューレット・パッカード，日立製作所などが，すでに同製品搭載のストレージ製品の販売を表明している．

また，コンピュータ・アソシエイツ，セイ・テクノロジーズ，ベリタスソフトウェアも，同製品対応のソフトウェアの販売を発表した．同製品は，NASなどのストレージ製品に最適化されたストレージ専用OSで，同社の「Windows Server 2003」をベースに開発されている．「Windows Server 2003」の機能でもある，VSS（Volume Shadow Copy Services）やVDS（Virtual Disk Service），DFS（Distributed File System），マルチノード・クラスタリング，マルチパスI/Oなどが搭載されており，NASデバイスをIPベースのSANに組み込んで管理できるようにする「iSCSI Software Initiator」にも対応している．



ストレージ業界で急成長を遂げるマクデータ

このように2003年の，iSCSI技術の本格的な市場参入がストレージ業界全体に及ぼした影響は，決して小さなものではない．今後，ネットワーク・ベンダやストレージ・ベンダでは，既存のFCだけではなく，iSCSI市場の動向も見据えた戦略が重要となってくるだろう．

では，実際にストレージ・ベンダでは，どのような戦略を考えているのだろうか．最後にコロラド州ブルームフィールドに拠点を構えるマクデータのSAN戦略について簡単に触れてみたい．

日本でストレージ関連のベンダというと，サン・マイクロシステムズやEMC，ブロケード，IBM，NEC，日立データシステムズといったメーカがすぐ頭に浮かんでくるが，マクデータという社名はなじみが薄いかもしれない．というのも現在，マ

クデータでは，IBM，デルコンピュータ，EMC，ヒューレット・パッカード，日立データシステムズ，NECなどのストレージ・パートナとの協力関係を重視した戦略をとっているためである．日本をはじめ中国，韓国，台湾，シンガポール，オーストラリア，ヨーロッパなど全世界に拠点をもち，約8,000ものデータ・センタにSANの導入，運用実績をもっている．

それでは現在，マクデータが構想するSAN戦略はどのようなものなのだろうか．同社の取締役会長であるジョン・ケリー氏（写真1）は，次のように語っている．約6年前から同社では，顧客のコストやビジネス・リスクを軽減するために「多機能ストレージ・ソリューション」に特化したプロバイダ・ビジネスを展開している．代表的なものとして，12/24/32ポートの中・小規模向けの「Sphereon（スフェリオン）4300/4500」スイッチ，64/140ポートのデータ・センタ・プラットホームなどの用途に向けた「Intrepid（イントレピッド）6064/6140」ダイレクタなどのハードウェア製品群を提供している．また，ソフトウェア製品群としては，エンド・トゥ・エンドでネットワーク管理を行う「SANavigator」や「EFCM（Enterprise Fabric Connectivity Manager）」などがある．これらのハードウェア/ソフトウェア製品群により，各地に分散するSAN環境下で活用されているマルチベンダ・プラットホームの統合を可能にしているという．同社のソリューションを採用することにより，オラクルでは全世界40か所に点在していたデータ・センタを，2か所に集約することに成功したという．

マクデータ製品の導入，運用実績は，全世界で約8,000のデータ・センタで採用されていると先に触れたが，現在はこれらのSANのすべてが，FCで接続されているという．おもにIBMやEMC，日立データシステムズのストレージを，同社のイントレピッドやスフェリオンで接続し，SANavigatorで管理するという運用が一般的のようだ．現在，これらのデータ・センタでは，FCからIPへの移行も考えられているという．ジョン・ケリー氏によれば，現在FC市場が30～50億ドル，IP市場が40～50億ドルとなっており，今後はFCとIPがSANの柱となってくるだろうとのことである．とくにIPによるストレージ・ネットワーキングについては，iSCSIをはじめiFCIP，FCIPなどが主流になりつつあるとみている．

次にSAN環境のIPへの移行に関しては，マクデータとしてどのような戦略を立てているのであろうか．同社では現在，顧客のデータ・インフラの中に「拡張性」，「インテリジェンス」，「インターネットワーキング」という三つの方向性を見出しているという．拡張性ではより多くのポート数，より大きなファブリック，より多くのエンド・デバイスなどが必要とされていることを示しており，インテリジェンスでは，より効率的な一元管理の重要性が問われているという．そしてインターネットワーキングでは，個別のSANアイランド注2の論理的な統合がキーポイントになってくるものと考えている．このような顧客の方向性に十全かつ迅速に応えるために，同社は2003年，ニシャンシステムズとサネラシステムズの2社を買収した．サネラシステムズは拡張性，ニシャンシステムズはインターネットワーキングの分野で，同社のビジネス戦略上，大きなメリットをもたらしたという．



写真1　マクデータ社 取締役会長 ジョン・ケリー氏

マクデータがSANを導入したデータ・センタには，すべて障害時の修復機能がサポートされており，これらの機能はニシャンシステムズの技術なのだという．ニシャンシステムズの技術には，「長距離サポート」，「iSCSIによるサーバの統合」，「SANルーティング」という三つの大きな特徴がある．今回の買収によってマクデータ製品となった，4ポート・スイッチ「Eclipse1620」では，マクデータやブロケードのSANに，1,000台のサーバをIPおよびFC経由で接続することができる．ニシャンシステムズは，買収当時で，すでに130社以上のユーザと500の稼動中のデバイスを持っていた．これに加えて2003年10月以降，約2か月弱で250万ドル分の受注をマクデータにもたらした．この事実からもジョン・ケリー氏は，ニシャンシステムズの技術により，今後も高い成長率が期待できるという感触を得たとしている．

マクデータは，2004年以降本格的なアジア・太平洋地域での事業展開も表明している．その第一段階が，ポール・ラス氏のアジア・太平洋地域担当ゼネラルマネージャ兼副社長への起用であろう．ポール・ラス氏は，EMCの中国・ASEAN担当のゼネラル・マネージャを長年努め，アジア・太平洋地域における技術系事業で20年以上にわたり指導的な役割を果たしてきたという．今後3年間で，アジア・太平洋地域におけるマクデータ製品の市場占有率を大幅に拡大させることを目標にしており，もっとも期待している地域として中国を，2番目に拡大が見込める地域として日本を挙げている．

注2：SANのマシンが各所に点在している現象．

きたむら・としゆき



第15回
GCCにおけるマルチスレッドへの対応

岸 哲夫

昨年12月にリリースされたLinuxのカーネル2.6は従来までのカーネルと比較して，大きく進化した．中でもカーネル・レベル・プリエンプションによる割り込み応答性能の向上については，とくに期待が大きい．マルチスレッドに関する機能の強化とともに，Linuxを利用していくうえで嬉しい進化だ．

今回はGCCにおけるマルチスレッド対応について検証してみる．　　(筆者)

最近のPentium4は，ほとんどWillametteコアです．つまり疑似的にマルチプロセッサとして使用できます．そしてLinuxも，昨年12月にリリースされたカーネル2.6ではマルチプロセッシングに関する機能がより強化されています．

とくにカーネル・レベル・プリエンプションの強化によって，大きく機能が進化しました．スケジューラは，プロセスとスレッドをまったく区別せずに扱います．従来のカーネルでは，待ち状態になっているプロセス(スレッド)を活性化したことでコンテキスト切り替えが発生したとき，システム・コールの出口でコンテキスト切り替えを行っていました．

つまり，割り込み処理の際に，実行中のプロセスより優先度の高い待ちになっているプロセスを割り込みハンドラから活性化しても，それにともなうスケジューリング処理は割り込み発生前に実行していたシステム・コールが終了するまで遅延されてしまいました．

カーネル2.6では，割り込みに対する応答性能を改善する目的で，即時にスケジューリングを行います．これをカーネル・レベル・プリエンプションと呼びます．スケジューラによって各プロセスには動作可能な時間が与えられていますが，この時間が過ぎると強制的に別のプロセスに切り替えられます．そのような動作を「プリエンプション」と呼びます．

従来，多くのUNIX系のOSでは，プリエンプションが発生するのはユーザ・プログラムを実行しているときだけでした．カーネル内部の処理を行っているときは，明示的にカーネル内で指定しない限り，プロセスのスケジューリングは発生しないようにプログラムが作られていました．

しかし，昨今のCPUの高性能化によって，動画のエンコード処理もPCで楽にできるようになりました．これをリアルタイムで行うためには，1秒間で30フレームの画像を処理しなければなりません．しかもディスク・アクセスを行い，音声処理を行いながらです．そんなときに，場合によっては数十ms程度の遅延時間がかかるシステム・コールを処理してしまうと，その処理時間にCPUが引きずられてしまいます．そうなると，その間に取り込まなければならない映像を逃してしまうでしょう．

そこで，カーネル内部の処理を行っている最中でも，プリエンプションを許可することで，そのようなリアルタイム性を高めることができます．ただし，カーネルは内部でプリエンプションが発生することを考慮してプログラムされてはいません．

そこでカーネルはマルチプロセッサを使用するときに必須の「スピンロック」と呼ばれるしくみに目を付け，スピンロックを実行している間以外はカーネル内部でもプリエンプション可能とすることに成功したのです．

よってカーネル2.6はマルチプロセッサまたは疑似マルチプロセッサであるHT(Hyper Threading)テクノロジ対応CPUのもとで使用することで，すばらしい性能を引き出すことが可能になります．その際にはマルチスレッド処理が必須になります．

以上のような理由で，今回はGCCにおけるマルチスレッド処理の基本を説明していきます．

Linuxカーネル2.6の概要

カーネル2.6の際立った特徴は次のようなものです．

▶ マルチプロセッサ・システム対応の強化

カーネル2.6ではCPUが32個でも問題なく動作するそうです．

▶ マルチスレッド対応の強化

後述しますが，pthread_mutex_lockやpthread_mutex_unlockの処理を効率良く行うことができます．

スレッドをスリープさせる場合，実アドレスが刻々と変化します．ほかの処理が進むにつれて，ページングなどで変わってしまうからです．その実アドレスと，仮想アドレスの変換を行う関数が充実しました．「futexシステム・コール」と呼ばれるものです．この機能でスレッドの同期実時間が減少し，スレッドを用いたアプリケーションの性能が良くなるはずです．

▶ ネットワーク処理の効率が上がった

▶ ファイルI/Oの効率化

複数のバッファへの読み書きを行うreadv/writevのコードが更新され，効率の良い処理ができるようになりました．

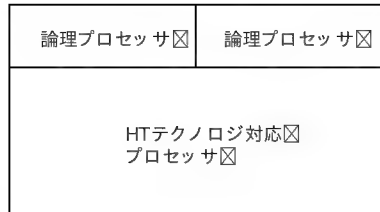


図1
HTテクノロジの概要

▶メモリ管理の効率化
▶対応するアーキテクチャの見直し
▶ACPIによる電源管理
▶対応デバイスの追加
▶ファイル・システムの機能追加

POSIX ACLがサポートされることになりました．これはWindowsにおけるアクセス制御に似ています．

▶ネットワーク・ファイル・システムの機能追加
▶LVMの強化
▶IPv6を実装するなどTCP/IP関連機能の強化

従来はUSAGIプロジェクトで作成されたカーネルでIPv6を実装してきましたが，オリジナルのカーネルで対応できるようになりました．

▶カーネル・レベル・プリエンプションの強化

この機能によって反応速度が速くなります．結果としてマルチスレッドが効果的に実行できます．

CPUを利用する際のスケジューリングが効果的に行えるため，根本的な処理速度の改善が可能になるはずです．

HTテクノロジの概要

図1のように32ビット論理プロセッサを2基装備した物理プロセッサという模式化が可能です．

具体的にどのようなリソースを共有しているかを説明します．

次のリソースに関しては，論理プロセッサごとに独自に持っています．

▶汎用レジスタ

この場合，論理プロセッサは32ビットCPUなので八つの32ビット汎用レジスタ（EAX，EBX，ECX，EDX，EBP，ESI，EDI，ESP）を装備しています．

▶コントロール・レジスタ

80286から存在しています．プロテクト・モードとリアル・モードの切り替えに使用したり，例外の発生をコントロールします．

▶割り込みコントロール用レジスタ

APIC（Advanced Programmable Interrupt Controller）と呼ばれています．

これはIntel社が1994年にリリースしたマルチプロセッサ向けのアーキテクチャ，およびそのための割り込みコントローラです．ユニプロセッサのPCではIRQは15までで，PIC（Programmable Interrupt Controller）で動作しています．これに対して，マルチプロセッサ対応のOSではAPICを使用しています．

一方，共有するリソースは以下のとおりです．

●キャッシュ
●実行ユニット
●分岐予測器
●コントロール・ロジック
●バス

これらを共有することで回路を簡略化し，消費電力を抑え，コストを低減しています．

（参考資料：Intel社の公式文書「ハイパー・スレッディング・テクノロジのアーキテクチャとマイクロアーキテクチャ」）

リスト1　プロセスの実行（test200.c）

```
/*
 *プロセスの実行
 */
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
      pid_t child_pid;
      child_pid = fork();
      execl("/usr/bin/xine","xine");
      system("ps -Al > test.log");
}
```

GCCを使ったマルチスレッド・プログラミングの基本

簡単にいえば，プロセスは固有のメモリ空間・固有のスタック上で動作し，スレッドは共有のメモリ空間・固有のスタック上で動作するものです．具体的な使用例をリスト1に示します．

まずはプロセスの場合です．システム・コールfork()で，親プロセスとまったく同じコピーを作成し，親には子のプロセスIDを，子にはゼロを返します．このコードでプロセスを生成できます．これは単純に別プロセスでxineを起動しているだけです（図2）．

親プロセスであるtest200のPIDは18727，子プロセスとして実行したxineは18728です．

一方，スレッドの場合はどうなるでしょうか？　POSIXスレッドを生成してみます．リスト2は単純にスレッドを実行するプログラムです．

通常は，このソースをコンパイルするには次のようにすればOKです．“-lpthread”でライブラリを指定します．

```
$ gcc -lpthread test201.c -o test201
$ ./test201
メインスレッドのスレッドID＝1074008704
スレッド1のスレッドID＝1082399936
スレッド2のスレッドID＝1090788416
```

```
F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  0  75   0    -   342 schedu ?        00:00:04 init
1 S     0     2     1  0  75   0    -     0 contex ?        00:00:00 keventd
1 S     0     3     1  0  75   0    -     0 schedu ?        00:00:00 kapmd
1 S     0     4     1  0  94  19    -     0 ksofti ?        00:00:00 ksoftirqd_C
1 S     0     9     1  0  85   0    -     0 bdflus ?        00:00:00 bdflush
1 S     0     5     1  0  75   0    -     0 schedu ?        00:00:02 kswapd
1 S     0     6     1  0  75   0    -     0 schedu ?        00:00:00 kscand/DMA
1 S     0     7     1  0  75   0    -     0 schedu ?        00:00:27 kscand/Norm
1 S     0     8     1  0  75   0    -     0 schedu ?        00:00:00 kscand/High
1 S     0    10     1  0  75   0    -     0 schedu ?        00:00:00 kupdated
1 S     0    11     1  0  85   0    -     0 md_thr ?        00:00:00 mdrecoveryd
1 S     0    15     1  0  75   0    -     0 end    ?        00:00:00 kjournald
1 S     0    72     1  0  85   0    -     0 end    ?        00:00:00 khubd
1 S     0  1833     1  0  75   0    -     0 end    ?        00:00:00 kjournald
1 S     0  2177     1  0  76   0    -   491 schedu ?        00:00:00 dhclient
5 S     0  2221     1  0  75   0    -   361 schedu ?        00:00:00 syslogd
5 S     0  2225     1  0  75   0    -   342 do_sys ?        00:00:00 klogd
5 S    32  2243     1  0  75   0    -   386 schedu ?        00:00:00 portmap
5 S    29  2262     1  0  85   0    -   382 schedu ?        00:00:00 rpc.statd
5 S     0  2328     1  0  84   0    -   341 schedu ?        00:00:00 apmd
5 S     0  2365     1  0  85   0    -   877 schedu ?        00:00:00 sshd
5 S     0  2379     1  0  76   0    -   508 schedu ?        00:00:00 xinetd
5 S    38  2396     1  0  75   0    -   598 schedu ?        00:00:00 ntpd
5 S     0  2422     1  0  75   0    -  1482 schedu ?        00:00:00 sendmail
1 S    51  2431     1  0  75   0    -  1428 pause  ?        00:00:00 sendmail
1 S     0  2441     1  0  85   0    -  4805 schedu ?        00:00:01 spamd
5 S     0  2450     1  0  75   0    -   352 schedu ?        00:00:00 gpm
1 S    49  2461     1  0  85   0    -  1289 schedu ?        00:00:00 jserver
1 S     1  2470     1  0  75   0    -   479 schedu ?        00:00:00 cannaserver
1 S     0  2482     1  0  75   0    -   356 schedu ?        00:00:00 crond
5 S    43  2604     1  0  75   0    -  1799 schedu ?        00:00:00 xfs
5 S     0  2613     1  0  75   0    -  1450 schedu ?        00:00:00 smbd
5 S     0  2617     1  0  75   0    -  1154 schedu ?        00:00:00 nmbd
1 S     2  2635     1  0  75   0    -   352 schedu ?        00:00:00 atd
4 S     0  2648     1  0  82   0    -   338 schedu tty1     00:00:00 mingetty
4 S     0  2649     1  0  82   0    -   338 schedu tty2     00:00:00 mingetty
4 S     0  2650     1  0  82   0    -   338 schedu tty3     00:00:00 mingetty
4 S     0  2651     1  0  82   0    -   338 schedu tty4     00:00:00 mingetty
4 S     0  2652     1  0  82   0    -   338 schedu tty5     00:00:00 mingetty
4 S     0  2653     1  0  82   0    -   338 schedu tty6     00:00:00 mingetty
4 S     0  2654     1  0  75   0    -  3747 schedu ?        00:00:00 gdm-binary
5 S     0  2684  2654  0  76   0    -  3989 wait4  ?        00:00:00 gdm-binary
4 S     0  2685  2684  0  76   0    - 16100 schedu ?        00:00:07 X
4 S     0  2724  2684  0  75   0    -  6084 schedu ?        00:00:00 gnome-sessi
1 S     0  2745  2724  0  84   0    -  1286 wait4  ?        00:00:00 Xsession
4 S     0  2747  2745  0  75   0    -  1214 schedu ?        00:00:00 kinput2
1 S     0  2786  2724  0  75   0    -   784 schedu ?        00:00:00 ssh-agent
0 S     0  2797     1  0  75   0    -  3092 schedu ?        00:00:00 gconfd-2
0 S     0  2799     1  0  75   0    -  1802 schedu ?        00:00:00 bonobo-acti
4 S     0  2801     1  0  75   0    -  4594 schedu ?        00:00:00 gnome-setti
4 S     0  2806  2379  0  75   0    -   664 schedu ?        00:00:00 fam
0 S     0  2813     1  0  75   0    -  4616 schedu ?        00:00:00 metacity
4 S     0  2817     1  0  75   0    -  6854 schedu ?        00:00:02 gnome-panel
4 S     0  2819     1  0  75   0    - 16582 schedu ?        00:00:02 nautilus
0 S     0  2821     1  0  75   0    -  4352 schedu ?        00:00:01 magicdev
0 S     0  2824     1  0  75   0    -  4458 schedu ?        00:00:00 eggcups
0 S     0  2826     1  0  75   0    -  3197 schedu ?        00:00:00 pam-panel-i
0 S     0  2828     1  0  75   0    -  6377 schedu ?        00:00:01 rhn-applet-
0 S     0  2829  2826  0  75   0    -   351 schedu ?        00:00:00 pam_timesta
0 S     0  2835     1  0  75   0    -  4521 schedu ?        00:00:00 notificatio
0 S     0  2844     1  0  75   0    -  6393 schedu ?        00:00:00 gnome-termi
4 S     0  2845  2844  0  83   0    -   461 schedu ?        00:00:00 gnome-pty-h
0 S     0  2846  2844  0  75   0    -  1326 wait4  pts/0    00:00:00 bash
5 S     0  2911  2613  0  75   0    -  1562 schedu ?        00:00:00 smbd
5 S     0  3264     1  0  94  19    -  2132 schedu ?        00:00:00 cupsd
4 S     0 18656  2379  0  75   0    -   396 schedu ?        00:00:00 in.telnetd
4 S     0 18657 18656  0  76   0    -   568 wait4  ?        00:00:00 login
4 S   500 18658 18657  0  75   0    -  1340 schedu pts/1    00:00:00 bash
4 S     0 18701  2846  0  81   0    -  1240 wait4  pts/0    00:00:00 su
4 S   500 18702 18701  1  78   0    -  1345 wait4  pts/0    00:00:00 bash
0 S   500 18727 18702  0  82   0    -   334 wait4  pts/0    00:00:00 test200
0 Z   500 18728 18727  3  84   0    -     0 do_exi pts/0    00:00:00 xine <defun
0 S   500 18729 18727  0  84   0    -  1274 wait4  pts/0    00:00:00 sh
0 R   500 18730 18729  6  85   0    -   789 -      pts/0    00:00:00 ps
```

図2 test200.log

```
スレッド 1を実行中
スレッド 2を実行中
スレッド 1を実行中
スレッド 2を実行中
スレッド 1を実行中
スレッド 2を実行中

$
```

このように別個のスレッド IDが付加されています．暗黙に実行されているメイン・スレッドに一つ，各スレッドに一つずつです．

リスト 3のコードで関数を実行するものが生成するアセンブラ・コードと比較してみましょう(リスト 4，リスト 5)．関数では「call thread_function1」で呼び出していますが，スレッド処理では thread_function1 のアドレスを pthread_create に渡しています．

次に「スレッドは共有のメモリ空間を使用する」ことを実証してみます．リスト 6(pp.175-176)を参照してください．実行結果は次のとおりです．

```
$ gcc -lpthread test202.c -o test202
$ ./test202
メインスレッドのスレッド ID= 1074008704
スレッド 1のスレッド ID= 1082399936
スレッド 2のスレッド ID= 1090788416
スレッド 1を開始します　Gintの初期値は=
1000
スレッド 2を開始します　Gintの初期値は=
2000
スレッド 1が終了しました　Gintの値は=
17134
スレッド 2が終了しました　Gintの値は=
21333
$
```

計算結果はめちゃくちゃです．この処理はスレッド 1の優先度を上げているので先に終わりますが，スレッド 2の優先度を上げても同じことです．

リスト 7(p.176)で検証してみましょう．実行結果は次のと

リスト 2　スレッドの生成と実行(test200.log)

```
/*
 *スレッドの生成と実行
 */
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

void *thread_function1(void *arg)
{
    int i;
    printf("スレッド1 のスレッド ID= %d¥n",pthread_self());
    sleep(1);
    for ( i=0; i<3; i++ )
    {
        printf("スレッド1 を実行中¥n");
        sleep(1);
    }
    return NULL;
}
void *thread_function2(void *arg)
{
    int i;
    printf("スレッド2 のスレッド ID= %d¥n",pthread_self());
    sleep(1);
    for ( i=0; i<3; i++ )
    {
        printf("スレッド2 を実行中¥n");
        sleep(1);
    }
    return NULL;
}

int main(void)
{
    pthread_t thread1;
    pthread_t thread2;
    printf("メインスレッドのスレッド ID= %d¥n",pthread_self());
    sleep(1);
    if ( pthread_create( &thread1, NULL, thread_function1,NULL))
    {
        printf("スレッド 1生成に失敗¥n");
        abort();
    }
    if ( pthread_create( &thread2, NULL, thread_function2,NULL))
    {
        printf("スレッド 2生成に失敗¥n");
        abort();
    }
    if ( pthread_join ( thread1, NULL ) )
    {
        printf("スレッド 1と合流に失敗¥n");
        abort();
    }
    if ( pthread_join ( thread2, NULL ) )
    {
        printf("スレッド 2と合流に失敗¥n");
        abort();
    }
    exit(0);
}
```

リスト 3　単純な関数処理のソース(test201a.c)

```
/*
 *単純な関数
 */
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

void *thread_function1()
{
    int i;
    for ( i=0; i<3; i++ )
    {
        printf("関数1 を実行中¥n");
        sleep(1);
    }
    return NULL;
}
void *thread_function2()
{
    int i;
    for ( i=0; i<3; i++ )
    {
        printf("関数2 を実行中¥n");
        sleep(1);
    }
    return NULL;
}

int main(void)
{
    char    *thread1;
    char    *thread2;
    thread1 =   thread_function1();
    thread1 =   thread_function2();
    exit(0);
}
```

リスト 4 スレッド処理(test201.s)

```
    .file   "test201.c"
    .section    .rodata
.LC0:
    .string "¥245¥271¥245¥354¥245¥303¥245¥311¥243¥261¥244¥316
   ¥245¥271¥245¥354¥245¥303¥245¥311¥243¥311¥243¥304¥241¥341%d¥n"
.LC1:
    .string "¥245¥271¥245¥354¥245¥303¥245¥311¥243¥261¥244¥362
                                          ¥274¥302¥271¥324¥303¥346¥n"
    .text
.globl thread_function1
    .type   thread_function1, @function
thread_function1:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $24, %esp
    call    pthread_self
    movl    %eax, 4(%esp)
    movl    $.LC0, (%esp)
    call    printf
    movl    $1, (%esp)
    call    sleep
    movl    $0, -4(%ebp)
.L2:
    cmpl    $2, -4(%ebp)
    jle .L5
    jmp .L3
.L5:
    movl    $.LC1, (%esp)
    call    printf
    movl    $1, (%esp)
    call    sleep
    leal    -4(%ebp), %eax
    incl    (%eax)
    jmp .L2
.L3:
    movl    $0, %eax
    leave
    ret
    .size   thread_function1, .-thread_function1
    .section    .rodata
.LC2:
    .string "¥245¥271¥245¥354¥245¥303¥245¥311¥243¥262¥244¥316
   ¥245¥271¥245¥354¥245¥303¥245¥311¥243¥311¥243¥304¥241¥341%d¥n"
.LC3:
    .string "¥245¥271v245¥354¥245¥303¥245¥311¥243¥262¥244¥362
                                          ¥274¥302¥271¥324¥303¥346¥n"
    .text
.globl thread_function2
    .type   thread_function2, @function
thread_function2:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $24, %esp
    call    pthread_self
    movl    %eax, 4(%esp)
    movl    $.LC2, (%esp)
    call    printf
    movl    $1, (%esp)
    call    sleep
    movl    $0, -4(%ebp)
.L7:
    cmpl    $2, -4(%ebp)
    jle .L10
    jmp .L8
.L10:
    movl    $.LC3, (%esp)
    call    printf
    movl    $1, (%esp)
    call    sleep
    leal    -4(%ebp), %eax
    incl    (%eax)
    jmp .L7
.L8:
    movl    $0, %eax
    leave
    ret
    .size   thread_function2, .-thread_function2
    .section    .rodata
    .align 32
.LC4:
    .string "¥245¥341¥245¥244¥245¥363¥245¥271¥245¥354¥245¥303
   ¥245¥311¥244¥316¥245¥271¥245¥354¥245¥303¥245¥311¥243¥311¥243
                                               ¥304¥241¥341%d¥n"
.LC5:
    .string "¥245¥271¥245¥354¥245¥303¥245¥3111¥300¥270¥300¥256
                                      ¥244¥313¥274¥272¥307¥324¥n"
.LC6:
    .string "¥245¥271¥245¥354¥245¥303¥245¥3112¥300¥270¥300¥256
                                      ¥244¥313¥274¥272¥307¥324¥n"
.LC7:
    .string "¥245¥271¥245¥354¥245¥303¥245¥3111¥244¥310¥271¥347
                              ¥316¥256¥244¥313¥274¥272¥307¥324¥n"
.LC8:
    .string "¥245¥271¥245¥354¥245¥303¥245¥3112¥244¥310¥271¥347
                              ¥316¥256¥244¥313¥274¥272¥307¥324¥n"
    .text
.globl main
    .type   main, @function
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $24, %esp
    andl    $-16, %esp
    movl    $0, %eax
    subl    %eax, %esp
    call    pthread_self
    movl    %eax, 4(%esp)
    movl    $.LC4, (%esp)
    call    printf
    movl    $1, (%esp)
    call    sleep
    movl    $0, 12(%esp)
    movl    $thread_function1, 8(%esp)
    movl    $0, 4(%esp)
    leal    -4(%ebp), %eax
    movl    %eax, (%esp)
    call    pthread_create
    testl   %eax, %eax
    je  .L12
    movl    $.LC5, (%esp)
    call    printf
    call    abort
.L12:
    movl    $0, 12(%esp)
    movl    $thread_function2, 8(%esp)
    movl    $0, 4(%esp)
    leal    -8(%ebp), %eax
    movl    %eax, (%esp)
    call    pthread_create
    testl   %eax, %eax
    je  .L13
    movl    $.LC6, (%esp)
    call    printf
    call    abort
.L13:
    movl    $0, 4(%esp)
    movl    -4(%ebp), %eax
    movl    %eax, (%esp)
    call    pthread_join
    testl   %eax, %eax
    je  .L14
    movl    $.LC7, (%esp)
    call    printf
    call    abort
.L14:
    movl    $0, 4(%esp)
    movl    -8(%ebp), %eax
    movl    %eax, (%esp)
    call    pthread_join
    testl   %eax, %eax
    je  .L15
    movl    $.LC8, (%esp)
    call    printf
    call    abort
.L15:
    movl    $0, (%esp)
    call    exit
    .size   main, .-main
    .ident  "GCC: (GNU) 3.3"
```

リスト 5　関数処理(test201a.s)

```
    .file   "test201a.c"
    .section    .rodata
.LC0:
    .string "¥264¥330¥277¥364¥243¥261¥244¥362¥274¥302¥271¥324
                                                        ¥303¥346¥n"
    .text
.globl thread_function1
    .type   thread_function1, @function
thread_function1:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    movl    $0, -4(%ebp)
.L2:
    cmpl    $2, -4(%ebp)
    jle .L5
    jmp .L3
.L5:
    movl    $.LC0, (%esp)
    call    printf
    movl    $1, (%esp)
    call    sleep
    leal    -4(%ebp), %eax
    incl    (%eax)
    jmp .L2
.L3:
    movl    $0, %eax
    leave
    ret
    .size   thread_function1, .-thread_function1
    .section    .rodata
.LC1:
    .string "¥264¥330¥277¥364¥243¥262¥244¥362¥274¥302¥271¥324
                                                        ¥303¥346¥n"
    .text
.globl thread_function2
    .type   thread_function2, @function
thread_function2:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    movl    $0, -4(%ebp)
.L7:
    cmpl    $2, -4(%ebp)
    jle .L10
    jmp .L8
.L10:
    movl    $.LC1, (%esp)
    call    printf
    movl    $1, (%esp)
    call    sleep
    leal    -4(%ebp), %eax
    incl    (%eax)
    jmp .L7
.L8:
    movl    $0, %eax
    leave
    ret
    .size   thread_function2, .-thread_function2
.globl main
    .type   main, @function
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $24, %esp
    andl    $-16, %esp
    movl    $0, %eax
    subl    %eax, %esp
    call    thread_function1
    movl    %eax, -4(%ebp)
    call    thread_function2
    movl    %eax, -4(%ebp)
    movl    $0, (%esp)
    call    exit
    .size   main, .-main
    .ident  "GCC: (GNU) 3.3"
```

リスト 6　共有のメモリ空間を使用する(test202.c)

```
/*
 * 共有のメモリ空間を使用する
 * 優先度: スレッド1 →スレッド2
 */
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sched.h>

int Gint;
void *thread_function1(void *arg)
{
    int i;
    int x;
    printf("スレッド1 のスレッド ID = %d¥n",pthread_self());
    for ( x=0; x<10000000; x++ )
    {
    }
    Gint    =   1000;
    printf("スレッド1 を開始します  Gint の初期値は=¥n%d¥n",Gint);
    for ( x=0; x<10000; x++ )
    {
        Gint++;
        for ( i=0; i<2000; i++ )
        {
        }
    }
    printf("スレッド1 が終了しました  Gint の値は=¥n%d¥n",Gint);
    return NULL;
}
void *thread_function2(void *arg)
{
    int i;
    int x;
    printf("スレッド2 のスレッド ID = %d¥n",pthread_self());
    for ( x=0; x<10000000; x++ )
    {
    }
    Gint    =   2000;
    printf("スレッド2 を開始します  Gint の初期値は=¥n%d¥n",Gint);
    for ( x=0; x<10000; x++ )
    {
        Gint++;
        for ( i=0; i<2000; i++ )
        {
        }
    }
    printf("スレッド2 が終了しました  Gint の値は=¥n%d¥n",Gint);
    return NULL;
}

int main(void)
{
    int i;
    pthread_attr_t  thread1_attr;
    pthread_attr_t  thread2_attr;
    struct  sched_param thread1_param;
    struct  sched_param thread2_param;
    pthread_t thread1;
    pthread_t thread2;
    printf("メインスレッドのスレッド ID = %d¥n",pthread_self());
    pthread_attr_init (&thread1_attr);
    pthread_attr_getschedparam (&thread1_attr, &thread1_param);
    thread1_param.sched_priority =  10000;
    pthread_attr_setschedparam (&thread1_attr, &thread1_param);
    if ( pthread_create( &thread1, &thread1_attr,
                                    thread_function1, NULL) )
    {
        printf("スレッド 1 生成に失敗¥n");
        abort();
    }
    pthread_attr_init (&thread2_attr);
    pthread_attr_getschedparam (&thread2_attr, &thread2_param);
    thread2_param.sched_priority =  0;
    pthread_attr_setschedparam (&thread2_attr, &thread2_param);
    if ( pthread_create( &thread2, &thread2_attr,
                                    thread_function2, NULL) )
    {
```

リスト 6　共有のメモリ空間を使用する(test202.c)(つづき)

```
        printf("スレッド 2 生成に失敗¥n");
        abort();
    }
    if ( pthread_join ( thread1, NULL ) )
    {
        printf("スレッド 1 と合流に失敗¥n");
        abort();
    }
    if ( pthread_join ( thread2, NULL ) )
    {
        printf("スレッド 2 と合流に失敗¥n");
        abort();
    }

    exit(0);

}
```

おりです．

```
$ ./test203
メインスレッドのスレッド ID＝1074008704
スレッド 1 のスレッド ID＝1082399936
スレッド 2 のスレッド ID＝1090788416
スレッド 1 を開始します　Gint の初期値は＝
1000
スレッド 2 を開始します　Gint の初期値は＝
2000
スレッド 2 が終了しました　Gint の値は＝
21654
スレッド 1 が終了しました　Gint の値は＝
21866
$
```

この処理はスレッド 2の優先度を上げているので先に終わりましたが，計算結果がおかしいのは同じことです．

● mutex の使用

それを防止するために変数 Gint を mutex を使用してロックします．具体的なコーディングはリスト 8のようになります．実行結果は次のとおりです．

```
$ gcc -lpthread test204.c -o test204
$ ./test204
```

リスト 7　共有のメモリ空間を使用する(test203.c)

```
/*
 * 共有のメモリ空間を使用する
 * 優先度: スレッド 2 →スレッド 1
 */
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sched.h>

int Gint;
void *thread_function1(void *arg)
{
    int i;
    int x;
    printf("スレッド1 のスレッド ID＝ %d¥n",pthread_self());
    for ( x=0; x<10000000; x++ )
    {
    }
    Gint    =   1000;
    printf("スレッド1 を開始します　Gint の初期値は＝¥n%d¥n",Gint);
    for ( x=0; x<10000; x++ )
    {
        Gint++;
        for ( i=0; i<2000; i++ )
        {
        }
    }
    printf("スレッド1 が終了しました　Gint の値は＝¥n%d¥n",Gint);
    return NULL;
}
void *thread_function2(void *arg)
{
    int i;
    int x;
    printf("スレッド2 のスレッド ID＝ %d¥n",pthread_self());
    for ( x=0; x<10000000; x++ )
    {
    }
    Gint    =   2000;
    printf("スレッド2 を開始します　Gint の初期値は＝¥n%d¥n",Gint);
    for ( x=0; x<10000; x++ )
    {
        Gint++;
        for ( i=0; i<2000; i++ )
        {
        }
    }
    printf("スレッド2 が終了しました　Gint の値は＝¥n%d¥n",Gint);
    return NULL;
}

int main(void)
{
    int i;
    pthread_attr_t  thread1_attr;
    pthread_attr_t  thread2_attr;
    struct  sched_param thread1_param;
    struct  sched_param thread2_param;
    pthread_t thread1;
    pthread_t thread2;
    printf("メインスレッドのスレッド ID＝ %d¥n",pthread_self());
    pthread_attr_init (&thread1_attr);
    pthread_attr_getschedparam (&thread1_attr, &thread1_param);
    thread1_param.sched_priority =  0;
    pthread_attr_setschedparam (&thread1_attr, &thread1_param);
    if ( pthread_create( &thread1, &thread1_attr,
                                         thread_function1, NULL) )
    {
        printf("スレッド 1 生成に失敗¥n");
        abort();
    }
    pthread_attr_init (&thread2_attr);
    pthread_attr_getschedparam (&thread2_attr, &thread2_param);
    thread2_param.sched_priority =  10000;
    pthread_attr_setschedparam (&thread2_attr, &thread2_param);
    if ( pthread_create( &thread2, &thread2_attr,
                                         thread_function2, NULL) )
    {
        printf("スレッド 2 生成に失敗¥n");
        abort();
    }
    if ( pthread_join ( thread1, NULL ) )
    {
        printf("スレッド 1 と合流に失敗¥n");
        abort();
    }
    if ( pthread_join ( thread2, NULL ) )
    {
        printf("スレッド 2 と合流に失敗¥n");
        abort();
    }

    exit(0);

}
```

```
メインスレッドのスレッド ID＝1074008704
スレッド1のスレッド ID＝1082399936
スレッド1を開始します　Gintの初期値は＝
1000
スレッド1が終了しました　Gintの値は＝
11000
スレッド2のスレッド ID＝1090788416
スレッド2を開始します　Gintの初期値は＝
2000
スレッド2が終了しました　Gintの値は＝
12000
$
```

この場合，スレッド2の優先度が高くても先にGintを取ってロックしたほうが優先になります．よってスレッド1が先に実行されます．

早い者勝ちでつかんだ変数をロックして，処理が終わったらアンロックしています．これがmutexを使用した排他処理です．計算結果は意図した値が保持されています．

スレッドのメモリ空間はプロセス内で共有されています．そのため静的な変数は同じプロセスに属するすべてのスレッドから操作可能となります．自動変数はスタック上にとられるためスレッドごとに固有に確保されることになります．

メモリの確保をスタックにしたい場合，スレッド処理ではallocaを使います．その際はオプションの指定でスタック・オーバフローをチェックしましょう．-fstack-checkを指定します．これは連載第5回で説明しています．

また，各スレッドが同じメモリ空間を共有しているということは，複数のスレッドから使用する変数の操作を行う場合には注意しなくてはなりません．そこでいちばん使われる手法が，先に述べたmutexを使用してロックすることです．

● スレッド実行権限の譲渡

mutexを使用しなくても，リスト9のような処理もできま

リスト8　mutexを使用して変数をロックする(test204.c)

```
/*
 *共有のメモリ空間を使用する
 *mutex使用
 */
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sched.h>
pthread_mutex_t test_mutex=PTHREAD_MUTEX_INITIALIZER;

int Gint;
void *thread_function1(void *arg)
{
    int i;
    int x;
    pthread_mutex_lock(&test_mutex);
    printf("スレッド1 のスレッド ID = %d¥n",pthread_self());
    for ( x=0; x<10000000; x++ )
    {
    }
    Gint    =   1000;
    printf("スレッド1 を開始します  Gintの初期値は=¥n%d¥n",Gint);
    for ( x=0; x<10000; x++ )
    {
        Gint++;
        for ( i=0; i<2000; i++ )
        {
        }
    }
    printf("スレッド1 が終了しました  Gintの値は=¥n%d¥n",Gint);
    pthread_mutex_unlock(&test_mutex);
    return NULL;
}
void *thread_function2(void *arg)
{
    int i;
    int x;
    pthread_mutex_lock(&test_mutex);
    printf("スレッド2 のスレッド ID = %d¥n",pthread_self());
    for ( x=0; x<10000000; x++ )
    {
    }
    Gint    =   2000;
    printf("スレッド2 を開始します  Gintの初期値は=¥n%d¥n",Gint);
    for ( x=0; x<10000; x++ )
    {
        Gint++;
        for ( i=0; i<2000; i++ )
        {
        }
    }
    printf("スレッド2 が終了しました  Gintの値は=¥n%d¥n",Gint);
    pthread_mutex_unlock(&test_mutex);
    return NULL;
}

int main(void)
{
    int i;
    pthread_attr_t  thread1_attr;
    pthread_attr_t  thread2_attr;
    struct  sched_param thread1_param;
    struct  sched_param thread2_param;
    pthread_t thread1;
    pthread_t thread2;
    printf("メインスレッドのスレッド ID = %d¥n",pthread_self());
    pthread_attr_init (&thread1_attr);
    pthread_attr_getschedparam (&thread1_attr, &thread1_param);
    thread1_param.sched_priority =  0;
    pthread_attr_setschedparam (&thread1_attr, &thread1_param);
    if ( pthread_create( &thread1, &thread1_attr,
                                  thread_function1, NULL) )
    {
        printf("スレッド 1生成に失敗¥n");
        abort();
    }
    pthread_attr_init (&thread2_attr);
    pthread_attr_getschedparam (&thread2_attr, &thread2_param);
    thread2_param.sched_priority =  10000;
    pthread_attr_setschedparam (&thread2_attr, &thread2_param);
    if ( pthread_create( &thread2, &thread2_attr,
                                  thread_function2, NULL) )
    {
        printf("スレッド 2生成に失敗¥n");
        abort();
    }
    if ( pthread_join ( thread1, NULL ) )
    {
        printf("スレッド 1と合流に失敗¥n");
        abort();
    }
    if ( pthread_join ( thread2, NULL ) )
    {
        printf("スレッド 2と合流に失敗¥n");
        abort();
    }

    exit(0);

}
```

リスト 9 スレッド実行権限の譲渡(test205.c)

```
/*
 * 共有のメモリ空間を使用する
 * スレッド実行権限の譲渡
 */
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sched.h>
pthread_mutex_t test_mutex=PTHREAD_MUTEX_INITIALIZER;

int Gint;
void *thread_function1(void *arg)
{
    int i;
    int x;
    sched_yield();
    printf("スレッド1 のスレッド ID = %d¥n",pthread_self());
    for ( x=0; x<10000000; x++ )
    {
        sched_yield();
    }
    Gint    =   1000;
    printf("スレッド1 を開始します  Gint の初期値は=¥n%d¥n",Gint);
    for ( x=0; x<10000; x++ )
    {
        Gint++;
        for ( i=0; i<2000; i++ )
        {
        }
    }
    printf("スレッド1 が終了しました  Gint の値は=¥n%d¥n",Gint);
    return NULL;
}
void *thread_function2(void *arg)
{
    int i;
    int x;
    printf("スレッド2 のスレッド ID = %d¥n",pthread_self());
    for ( x=0; x<10000000; x++ )
    {
    }
    Gint    =   2000;
    printf("スレッド2 を開始します  Gint の初期値は=¥n%d¥n",Gint);
    for ( x=0; x<10000; x++ )
    {
        Gint++;
        for ( i=0; i<2000; i++ )
        {
        }
    }
    printf("スレッド2 が終了しました  Gint の値は=¥n%d¥n",Gint);
    sched_yield();
    return NULL;
}

int main(void)
{
    int i;
    pthread_attr_t  thread1_attr;
    pthread_attr_t  thread2_attr;
    struct  sched_param thread1_param;
    struct  sched_param thread2_param;
    pthread_t thread1;
    pthread_t thread2;
    printf("メインスレッドのスレッド ID = %d¥n",pthread_self());
    pthread_attr_init (&thread1_attr);
    pthread_attr_getschedparam (&thread1_attr, &thread1_param);
    thread1_param.sched_priority =  1000;
    pthread_attr_setschedparam (&thread1_attr, &thread1_param);
    if ( pthread_create( &thread1, &thread1_attr,
                                  thread_function1, NULL) )
    {
        printf("スレッド 1 生成に失敗¥n");
        abort();
    }
    pthread_attr_init (&thread2_attr);
    pthread_attr_getschedparam (&thread2_attr, &thread2_param);
    thread2_param.sched_priority =  1000;
    pthread_attr_setschedparam (&thread2_attr, &thread2_param);
    if ( pthread_create( &thread2, &thread2_attr,
                                  thread_function2, NULL) )
    {
        printf("スレッド 2 生成に失敗¥n");
        abort();
    }
    if ( pthread_join ( thread1, NULL ) )
    {
        printf("スレッド 1と合流に失敗¥n");
        abort();
    }
    if ( pthread_join ( thread2, NULL ) )
    {
        printf("スレッド 2と合流に失敗¥n");
        abort();
    }

    exit(0);

}
```

す．実行結果は次のとおりです．

```
gcc -lpthread test205.c -o test205
$ ./test205
メインスレッドのスレッド ID= 1074008704
スレッド 1のスレッド ID= 1082399936
スレッド 2のスレッド ID= 1090788416
スレッド 2を開始します  Gint の初期値は=
2000
スレッド 2が終了しました  Gint の値は=
12000
スレッド 1を開始します  Gint の初期値は=
1000
スレッド 1が終了しました  Gint の値は=
11000
```

この場合，結果こそ正しくなっていますが，このような処理では無理があると思います．sched_yield()を実行するたびに他スレッドに実行権限を渡します．結果，自スレッドは実質的にスリープしています．

● セマフォの使用

セマフォと呼ばれるプログラミング手法があります．もともとはプロセス間の実行順序などを管理するためにあるものですが，しっかり管理しないと混乱を招きます．多数の開発者がいるプロジェクトでは，管理者が管理しなければまともなシステムが構築できなくなります．

スレッドの場合は，同一プロセス内に限って使用できるセマフォを使うことで，閉じている状態にできます．リスト 10のような方法で実現できます．実行結果は次のとおりです．

```
$ gcc -lpthread test206.c -o test206
$ ./test206
メインスレッドのスレッド ID= 1074008704
スレッド 2のスレッド ID= 1090788416
スレッド 2を開始します  Gint の初期値は=
2000
```

リスト 10　セマフォを使ったスレッド管理（test206.c）

```
/*
 * 共有のメモリ空間を使用する
 * セマフォを使ったスレッド管理
 */
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sched.h>
#include <semaphore.h>


int Gint;
sem_t sem;
int count;
void *thread_function1(void *arg)
{
    int i;
    int x;
    int ret;
//セマフォの値がゼロ以上になるまでスレッドを停止
    ret = sem_wait(&sem);
    printf("スレッド1 のスレッド ID = %d¥n",pthread_self());
    for ( x=0; x<10000000; x++ )
    {
    }
    Gint    =   1000;
    printf("スレッド1 を開始します  Gint の初期値は=¥n%d¥n",Gint);
    for ( x=0; x<10000; x++ )
    {
        Gint++;
        for ( i=0; i<2000; i++ )
        {
        }
    }
    printf("スレッド1 が終了しました  Gint の値は=¥n%d¥n",Gint);
    return NULL;
}
void *thread_function2(void *arg)
{
    int i;
    int x;
    int ret;
    printf("スレッド2 のスレッド ID = %d¥n",pthread_self());
    for ( x=0; x<10000000; x++ )
    {
    }
    Gint    =   2000;
    printf("スレッド2 を開始します  Gint の初期値は=¥n%d¥n",Gint);
    for ( x=0; x<10000; x++ )
    {
        Gint++;
        for ( i=0; i<2000; i++ )
        {
        }
    }
    printf("スレッド2 が終了しました  Gint の値は=¥n%d¥n",Gint);
//セマフォの値に1 を加算
    ret = sem_post(&sem);
    return NULL;
}

int main(void)
{
    int i;
    int ret;
    pthread_attr_t  thread1_attr;
    pthread_attr_t  thread2_attr;
    struct  sched_param thread1_param;
    struct  sched_param thread2_param;
    pthread_t thread1;
    pthread_t thread2;
//このプロセスだけで使用できるセマフォ
    count   =   0;
    ret = sem_init(&sem, 0, count);
//
    printf("メインスレッドのスレッド ID = %d¥n",pthread_self());
    pthread_attr_init (&thread1_attr);
    pthread_attr_getschedparam (&thread1_attr, &thread1_param);
    thread1_param.sched_priority =  1000;
    pthread_attr_setschedparam (&thread1_attr, &thread1_param);
    if ( pthread_create( &thread1, &thread1_attr,
                                  thread_function1, NULL) )
    {
        printf("スレッド 1 生成に失敗¥n");
        abort();
    }
    pthread_attr_init (&thread2_attr);
    pthread_attr_getschedparam (&thread2_attr, &thread2_param);
    thread2_param.sched_priority =  1000;
    pthread_attr_setschedparam (&thread2_attr, &thread2_param);
    if ( pthread_create( &thread2, &thread2_attr,
                                  thread_function2, NULL) )
    {
        printf("スレッド 2 生成に失敗¥n");
        abort();
    }
    if ( pthread_join ( thread1, NULL ) )
    {
        printf("スレッド 1と合流に失敗¥n");
        abort();
    }
    if ( pthread_join ( thread2, NULL ) )
    {
        printf("スレッド 2と合流に失敗¥n");
        abort();
    }

    exit(0);

}
```

```
スレッド 2が終了しました  Gintの値は=
12000
スレッド 1のスレッド ID = 1082399936
スレッド 1を開始します  Gintの初期値は=
1000
スレッド 1が終了しました  Gintの値は=
11000

$
```

結果も正しく，安全確実にスレッドの実行順序をコントロールできました．

ほかにも条件変数を設定し，シグナルや時刻指定でスレッドの実行を管理する方法があります．コードを実装すれば，容易にネットワークからのスレッドの呼び起こし，そのほかキーボードやマウス・イベントでも可能になります．

おわりに

このようにGCCでもまったく問題なくマルチスレッド処理ができることがわかったと思います．

現実には実プロセッサが複数なければ，カーネルのタイムシェアで疑似的にマルチスレッドが実行されているだけですが，カーネル2.6を使い，Xeonプロセッサを複数使ったPCで，マルチスレッド・プログラミングを行えば，より効率の良いコードが書けることと思います．

次回は「最適化オプション」の補足を行います．

きし・てつお

シニアエンジニアの技術草子 参拾八之段

◆時は移れど

旭　征佑

● 前世紀最後の大発明

筆者が学生のころ，「今世紀中の開発は無理だ」といわれていたものがいくつかあった．その中でとくに記憶に残っていたものが，後に日亜化学が量産化に成功した青色発光ダイオードだ．開発されたのは1993年，現在は米カルフォルニア大学バーバラ校の教授となっている中村修二氏によるものだ．そんな氏が，日亜化学に対して裁判を起こし，200億円を勝ち取ったというニュースは，まだ記憶に新しい．

日本企業では，勤務時間中の発明などの特許は，会社の所有となると職務規定などで規定されていることが多い．しかし，本来ならば特許は個人が所有するはずだ．中村氏は，まず現在，会社が登記している青色発光ダイオードの特許権の一つに関し，会社への特許権委譲を無効とする裁判を起こした．これは，2002年9月の中間判決で退けられた．そのため，裁判の争点は，中村氏に対する特許委譲の「相当の対価」がいくらになるか，という点に移っていた．

2003年1月30日，東京地裁は，相当の対価として200億円の支払いを命じたのだ．判決は対価を604億円としたが，氏の請求額が200億円だったので，この額が限度になった．どちらにせよ，裁判所が個人の研究成果に200億という対価の存在を認めたというのは衝撃的，かつ画期的なことだろう．産業界に与える影響も計り知れないものがある．中村氏の行動は今後の日本の研究開発者のためにも賞賛に値する．

このようなことは，今までの日本では考えられなかった．意外にも，筆者の身近にも例が二つほどあるので紹介しよう．

定年後を余裕で過ごしている知人は，かつて金属加工の職場で働いていた．彼は，職場で大量に発生する一塊の金属の削りカスをつかんで，油で汚れた鉄板をこすると汚れがよく落ちることに気がついた．さっそく商品化することになり，売れに売れた商品が現在も台所で活躍する「金属カールたわし」だ．

発明者である彼は，当然ながら会社からごほうびをもらえることになったそうだ．それは，ヨーロッパ旅行だったという．30～40年前のことだ．当時はたいへん高価だったのは事実だろう．しかし，会社が得た利益を考えるとそれだけではちょっとかわいそうな気がする．

もっと最近の話をしよう．筆者の友人は，あるスクリーン・エディタを開発した．上司に相談したが，会社で販売しても彼の懐には一銭も入らないという．良くて本部長か事業部長表彰の金一封だ．開発は自宅でやったという話だが，それを証明するのは難しい．少なくとも開発のきっかけは職場にあった．つまらない会社の判断に，結局，彼は許可をもらって会社を辞め，社外に販路を求めることになったのだ．

このエディタは，MS-DOS時代にすぐ有名になったエディタだ．その後も精力的に機能を追加し，すでに200万本近くが販売されているという．彼は，そこから得たロイヤリティに自己資金を加えて，東京近郊に一戸建てを購入したと聞いている．一般的なサラリーマンと同程度の収入なのかもしれない．もっとも，会社に残っていれば，給料以外にわずかな金額しか手に入れることはできなかっただろうが….

2人とも会社に対して強い主張もすることもない，つつましやかな日本人ということになる．

● 日本の研究開発の空洞化を招くのか

裁判の話に戻そう．先の判決には産業界も反発している．経済同友会代表の北城恪太郎代表幹事（IBM）は，「異常な判決」としたうえで，「企業に多大な負担を強いるなら日本で研究開発する意味がなくなり，空洞化を招く」，「製品化や販売などいろいろな人の努力があって利益が出るので，研究開発だけ多大な額というのはおかしい」と痛烈に批判している．

また，原告の日亜化学は「ノーリスクで終身雇用，あるいは安定収入という企業の中にあって，巨額のリスク負担をした企業に破天荒ともいえる巨額の成功報酬を請求することは，安定収入と巨額のリスク報酬の二重取りを求めるものであって理論上許されないことであり，もしそのような二重取りが認められれば日本企業の研究開発活動は成り立たない」などとして，ただちに控訴している．

どちらも内容はちょっと過激だ．産業界への配慮として強い表現にしたに違いないと信じたい．

一般的に，日本では，「社員の発明は会社の財産」という考えが蔓延している．これは明治の渋沢栄一氏が推し進めた殖産興業以来，日本独自の企業風土として定着している．新しい発明をしたとしても，それは会社の環境や資源を使用したものであり，それまでにはすでに会社が多大な投資をしていると考えるのだ．



シニアエンジニアの技術草子

また，画期的な開発や発明で会社の利益に多大な貢献をしたとしても，その分の見返りは，すでに会社の指示に基づいて働いた分の「給料として支払い済み」とされる．労働基準法でいう「労働の対償」は，すでに「賃金」として支払われていると解釈される．さらに対価を払うとなると二重払いであり，雇用主には支払う義務はないということになる．これらは，日本の産業が今までのように伸び続け，かつ終身雇用，年功序列を維持するためにも，とても重要な考えだった．

● 時代も，人も変わっている？

しかし，時代は変わりつつある．すでに右肩上がりの時代は終焉を迎えた．終身雇用も，年功序列も保証の限りではない．社員だって，企業に守られて生きていこうなどと思っちゃいない．「フリーター」ということばに代表されるように，少なくとも今の若い世代は，どんどん企業からフリーになっていく．

こんな時代だからこそ，技術開発が生命線の企業にとって，社内研究者の処遇が改めて問われようとしている．経済の国際化の中で，優秀な人材の流出を防ぐためにも，企業の知的財産権に対する意識改革が重要になってきていることを，日本企業は認識しなければならないはずなのだ．

ただし，日本企業にいては，中村氏のように大胆な行動に出ることのできる人はまだまだ少ないだろうと筆者は思う．日本企業も，それに付け込むに違いない．

少し前に，TVでおもしろい実験を見た．これには強烈な印象を受けた．その概要は，だいたい次のとおりだ．

18歳くらいの若い男女を4人ずつ2組に分けて，渋谷でアンケートを配らせるバイトをさせた．この時点で実験の趣旨は伝えていない．こんな簡単なことだが，たいへんおもしろい結果が出てしまった．

一つのチームは最初から路行く人に次々とアンケートを配っていく．アンケートを拒否する人も多いが，そんなこと気にもかけず，積極的に配っていく．顔には笑みがこぼれている．

もう一方のチームは，アンケートを何人かに拒否されてからは，動きが止まってしまった．路行く人の表情を伺うようになり，ほとんどアンケートを配れなくなってしまい，引きつった顔のまま，そろって路肩に立ちん坊になってしまった．

もちろん，男女，性格など別段区分けしたわけではない．配り方など指導したわけでもない．では，いったい何が違ったのか．



実は，次々と平気でアンケートを配ることができたのは，帰国子女のグループだったのだ．つまり，アンケートを断られても，気にせずに次々とにこやかにトライできたのが帰国子女だったというわけだ．一方，また断られるかもしれないと恐れてしまい，配ることができなくなってしまったのが，残念ながら我々日本人の気質という結論だった．最近の日本人は，変わったといわれる．しかし，この実験では，こんな若い人たちでも，今までの日本人と本質が何ら変わっていないことが証明されてしまったのだ．

グローバル社会などといって英語を学ばせ，実力主義の浸透，終身雇用の崩壊などといって会社は変わっていく．しかし，やはり日本人は簡単には変わらない．これからの企業は，今までのように社員を守ってはくれない．だからといって，中村氏のように企業に対して十分な対価を要求するという行動は，日本人の気質として，簡単にはできないのかもしれない．さらに，国際社会に飛び出す人も多いだろう．そこには，なおさら大きな試練が待っていそうだ．帰国子女たちがアンケートを渡す，溌剌（はつらつ）たる姿を思い出して，改めてそう思った．

あさひ・しょうすけ　テクニカル・ライタ
イラスト　森 祐子

Engineering Life in

フリー・エンジニアという仕事（第二部）

▌今回のゲストのプロフィール
ボブ・アイゼンスタッド（Bob Eisenstadt）：LSI設計エンジニアとして過去20年近くの経験を有する．VLSI Technology Inc.（現在はPhilipsの一部）でASIC設計の経験を積んだ後，Supermac, Radius, Silicon Graphics, 3DFX，Silicon Imageなどのグラフィックス関係の企業でLSI設計の外部スペシャリストとして活躍する．そのほかにスタート・アップの設立の経験や，パテント取得の経験を有する．オフには運動，造園や家族と過ごす．マサチューセッツ州ボストン出身．

前回まで

大学卒業後，シリコン・バレーに来たが，早い時期にレイオフされてしまった．この経験を活かしつつ老舗（しにせ）のASICベンダに就職し，ASICエンジニアとして仕事を続ける．その後，Silicon Graphicsでフリー・エンジニアとして働くきっかけができ，さまざまなシリコン・バレーの企業でフリー・エンジニアとして働く．

☆ ASICからCOTへの変化

トニー ボブさんはグラフィクス関係のデバイスを開発された経験が多いのですが，これには何か理由が？

ボブ フリー・エンジニアとしての仕事を，Silicon Graphicsからスタートしたからでしょうね．この分野の仕事をRadiusやSupermacで続け，その後は3DFXに行き，2001年にはSilicon Imageという会社でプロジェクトに参加しました．そして9.11で一気に景気が冷え込みましたが，景気の良いときは本当に仕事が多く，自分で選べる状態でした．

また，当時は主流が2Dから3Dにシフトするときだったので，グラフィクス関係にずっと携わっていられたのだと思います．インターネット・バブルの時期でしたが，同じくCOT（Customer Owned Tooling）が流行った時期でもあったので，それも私のできる仕事を増やしてくれていたと考えています．

トニー それまでのASIC開発は，顧客であるシステム屋さんやセット・メーカが論理設計まで行い，その後はASICベンダが1チップ化するなりデバイス化を行ってきましたが，COTになると，開発は論理設計以上にマスク開発まで顧客が担当することになりましたね．TSMCやCharter，IBMなどの純粋な半導体ファウンダリと直接やりとりする形になりました．これで今まで以上に半導体開発全体に携わったことのあるエンジニアが重宝されたのですね？

ボブ 完全にデバイス化をする仕事を自前で行い，すべてのコストをコントロールし，自社の知的財産をIP化してコントロールすることでシステム・メーカやセット・メーカは差別化を達成できると考えていたわけです．

しかし，マスク代や開発費が高騰しているので，コスト・メリットがどれぐらい出るのかは難しい判断だと思います．

トニー マスク代はよく聞く問題ですが，いくらくらいでしょうか？

ボブ 0.18μmプロセスで30万ドル（約3200万円），0.13μmで75万ドル（約8000万円），90nmで150万～170万ドル相当になります．ですから，1品種で1万個しか作らないデバイスはコスト的に無意味になってきています．

現状は，歩留まりが安定している0.18μmが主流のように感じますし，マスク代もしだいに安定してきています．0.15μmも0.18μmと似た歩留まりとコスト体系なので，この二つのプロセスは長く使われると思います．これ以上のパフォーマンス…大量のゲートを収めたいとか，パフォーマンスを要求しているデバイスなどは90nmを使うことになります．今後，e-beamの利用でさらにコストが下がってくると思います．

システム化においては，マルチチップ・モジュール（MCM）などがよく議論されています．しかし，結局使えるデバイスを採取するところで歩留まりが決まってくるので，いかに利用可能なデバイスを採取するかが大きな問題になり，コスト的に見合わないケースが多くあります．ですから，設計がしっかりしていれば1チップ化したほうが歩留まりを向上させることが可能なようで，今後もシステムを1チップ化していく方向は衰えないと思います．

☆ 仕様書を詰めることは難しい

トニー ずっと設計に携わっていましたが，とくに最近のトレンドなどはありますか？

ボブ 2.0μmがつい10年前まで使われていたプロセスで，それよりもさらにプロセス技術が発達しました．同じく設計技術も進化してきました．しかし，フロント設計，つまり論理設計やシステム設計と，バックエンド設計（レイアウト・レベルでの設計）がうまく分離できなくなっています．ゲート遅延以上に配線遅延をうまく見積もったり，モデリングすることが大切になっています．ここは，設計ツールが飛躍的に進化したので，自動化が進んでいると思います．

フロント系設計と論理設計では言語設計，つまりVerilog-HDLやVHDL，そして新しいSystemCを使った設計などが当たり前になってきています．また，バックエンド系のツールも最近はとても優秀で，指定したスペック内でなんとか配置配線をしようとがんばってくれます．ツールがたいへん進化しているので，実際のRTLから始まる設計については問題を感じません．私がもっとも問題だと感じているのは，デバイスがあまりにも大きくなっているので，一人のエンジニアが仕様のすべてを理解するのが不可能になっている点です．

トニー アーキテクトとか熟練のエンジニアがいても，仕様書が不完全ということでしょうか？

ボブ そうですね．シリコンバレーだと10名から20名のエンジニアに対して，1～3名のアーキテクトがいるという例が一般的です．アーキテクト達は，全体的なブロック図を描いた

り大まかなボトルネックになるポイントを指摘したり，かなり設計上の要点を押さえてくれます．おぼろげな輪郭から徐々に詳細な点を作り上げていきます．ですから，実際にRTLの設計段階に入ってシミュレーションしてみて，初めてFIFOの深さが足りないとかデータ・パスの遅延が長すぎるとかブロック間のタイミングが難しいといった問題がわかるのです．

トニー きちんとモデリングや仕様書を書いてもですか？

ボブ 一人の人間が理解できる範囲を超えているのでモデリングも難しいし，仕様もことばで表せないことがたくさんあります．また，アーキテクトとか熟練エンジニアはとても賢いのですが，すべてを紙に書くといった細かさに欠ける人が多いので，チーム全体に情報が伝わらない例が多くあります．

設計時間の多くは，自分にアサインされたブロックを中心としてチップ全体を理解しようとする，チーム全体の「勉強時間」に費やされる例がほとんどでしょう．

トニー よくある話ですが，納期が間に合わないのでとりあえずどんどんお客さんからもらった仕様書やアーキテクトから渡された仕様書で試行錯誤しながらRTLをガリガリ書いていく…そしてブロック間のシミュレーションを行ったときにやっと問題が見えてくるというやつですね．

ボブ まったくおっしゃるとおりで，どこも似たような手順で手探り状態でRTLを書き始めます．大幅な変更は当たり前とかね（苦笑）．しかし，おもしろいことに，パフォーマンス・モデルやトランザクション・モデルといったハイレベル・モデリングを徹底している会社は，最終的に良い結果を出しています．

トニー それはどういうことですか？

ボブ とくにグラフィクス関連の設計では，細かいチューニングによってパフォーマンスを最高にすることが可能です．しっかり手順を踏んでモデリングをしてきた会社が，チューニングの段階で何をしなければいけないのかをしっかりと把握しているので，開発の成果も確実に近くなります．また，システム全体の問題点の絞り込みもできています．だからモデリングなどに対して，チーム全体でじっくり取り組めればベストですね．

余談ですが，最近インドなどに設計の仕事が流れてしまうことを心配している人達がいますが，私はそうは思いません．しっかりと紙の仕様書で記述できるデバイスやブロックなどは可能だと思えるのですが，普通に行われているデバイス開発はどうしても徐々に作り込むプロセスが大切なので紙に書き残せる仕様書は少なく，システム関連の人達とデバイス開発の人達がいっしょに仕事ができる環境が必要です．こういう設計は距離や時差があると難しくなると思います．だから，海外の開発部隊…つまりインドなどに流せる仕事は，簡単なデバイスやブロックなどが妥当だと思います．

☆ 検証環境のアーキテクト


ボブ・アイゼンスタッド氏

トニー 理想的には本当にしっかりとモデリングすることがベストですが，実際にはなかなかできていないということですね…．これら以外に問題と感じている所は？

ボブ 仕様書レベルでのモデリングに関連する話ですが，検証のアーキテクトという考えかたが必要です．検証は後回しになっていますし，複雑なデバイスなので検証もとても複雑になります．同じ基礎アーキテクチャであれば過去の検証資産の流用などがおおいに可能になります．Intelなどが良い例です．基本アーキテクチャを進化させてゆく設計プロセスですから，検証チームも過去の資産の流用や，先回りして検証に必要なテスト・ベンチやテスト・ファイル，評価ボードなどを開発することが可能になります．

新しいアーキテクチャで設計するとゼロに近い状態から検証に必要な環境を作り出さなければならないので，それだけ費用と時間がかかります．場合によっては，これは設計サイクルに1～2年追加されることになってしまいます．ですから，ベンチャ企業でチップを一発で成功させるには，相当な努力やくふうが必要となります．逆に既存のベンダは検証の環境を流用するなどのコスト・メリットが得られます．ですから，仕様書レベルで設計の全体像を把握すると同時に，検証の環境をいかに設計していくかを把握する「検証のアーキテクト」が大切になります．

トニー ここでもハイレベル・モデリングが必要ですか？

ボブ しっかりしたモデリングがある場合，チップのボトルネックや問題点の把握ができているので，そこから検証やテストの段階で何をしなければいけないか，ある程度予測できます．グラフィックスの場合，FPGAなどのプロトタイピングによって実データを流さないとわからないことがたくさんあるので，ソフトウェア・シミュレーションだけに頼れない部分があるとか…．こういうのを洗い出して行くことがモデリングを行うことで，ある程度把握できるのです．全部とはいいません（苦笑）．少なくても私が見てきた企業で徹底的に差が出ていたのはこういうことができていた会社です．（次回に続く）

次回の予告：

フリー・エンジニアとしての生活などについて伺う．また今後のエンジニアリングについても話題が出た．

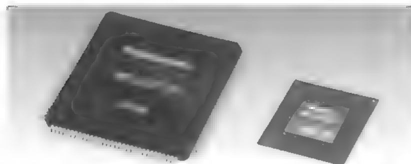
トニー・チン htchin@attglobal.net WinHawk Consulting

●ディジタルAV機器向け32ビット・マイコン

MN103SF77R MN103SF66R

- 0.18μmプロセスを用いた1Mバイトと大容量なフラッシュ・メモリを内蔵.
- MN103SF77Rはオーディオ機器や光ディスク機器向け，NN103SF66Rはディジタル・ビデオ・カメラ向け.
- 外部15MHz発振時，PLLにより4逓倍したクロックを内部クロックに使用し，内部60MHzの高速処理を120mW（60MHz/3.0V）～2mW/MHzで実現.
- 新規に開発した32ビット・コア「AM32L」を搭載.
- 最小レベルのメモリ・セルを使用することで，フラッシュ・メモリ搭載マイコンとして，高レベルの高集積化を実現.

●サンプル価格：¥1,600（MN103SF77R）
¥3,000（MN103SF66R）



■松下電器産業（株）
TEL：075-951-8151
E-mail：semiconpress@scd.mei.co.jp

●16ビット・マイコン

MSP430ファミリ

- 最大10Kバイトの大容量RAMを持ち，低消費電力を実現する，ミックスト・シグナル・マイコン.
- 8チャネルの12ビットA-Dコンバータ，および2チャネルの12ビットD-Aコンバータを搭載.
- I²Cインターフェース，10本のPWM出力，3チャネルのDMAコントローラをサポート.
- ミックスト・シグナル機能により，完全なフィードバック制御を1チップで実現.
- 搭載される拡張DMAは，すべて周辺機器からのトリガを装備しており，複雑かつ設定可能なデータ転送をCPUへの割り込み処理なしで実現でき，MCUベースの信号処理を大幅に加速.
- DMA転送のトリガ信号源をCPUから完全に独立させることができ，メモリ，内蔵ハードウェア，外付けハードウェア間で高精度な転送制御を実現.

●価格：$8.25～$8.95（1,000個時）

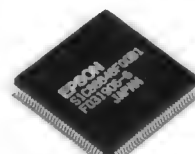
■日本テキサス・インスツルメンツ（株）
FAX：0120-81-0036
URL：http://www.tij.co.jp/pic/

●8ビット・マイコン

S1C88848

- リモート・コントローラを搭載しており，赤外線リモコンLEDとトランジスタを接続することで，各種リモコンを実現可能.
- スタンバイ時の消費電力1.7μAの低消費電力が特徴で，最大1,632ピクセルを駆動可能なドット・マトリクスLCDコントローラ/ドライバ，3種類のタイマ，調歩同期/クロック同期が選択可能なシリアル・インターフェースを内蔵.
- 表示付きバッテリ駆動のエアコン，テレビ，AV機器，ホーム・エンターテイメント機器などの家電機器リモコンに適する.
- リモコンに適したROM 48Kバイト，RAM 1.5Kバイトのメモリを搭載.
- リモコンへの日付日時表示を可能とする，時計用8ビット・タイマを内蔵.

●サンプル価格：
¥900



■セイコーエプソン（株）
TEL：042-587-5816
URL：http://www.epsondevice.com/

●オーディオ・ビデオ全二重CODEC

STm8000

- DVD，HDDレコーダ向けのオーディオ/ビデオ・エンコーディング回路に加え，DVDプレーヤの機能を提供.
- フルDVDレコーダ，DVR，コンバージェンス製品などに対し，機能セットを備えた費用対効果に優れたソリューションを提供.
- アナログ・テレビ録画，無料地上波ディジタル放送サポート，タイム・シフト，DV映像のディスク・ダビング，メモリ・カード読み込み，DivXデコード，DVDオーディオなどに対応.
- DVD±RW，DVD-RAMなど，すべてのDVDレコーダ・フォーマットに対応.
- 同製品をベースとした完全なDVDレコーダ・リファレンス・プラットホームを開発中.

●サンプル価格：¥2,900（10,000個時）



■STマイクロエレクトロニクス（株）
TEL：03-5783-8340 FAX：03-5783-8216

●マルチメディア・システムLSI

MP2520F

- MPEG-1/2/4，H.264，JPEG，MotionJPEGなど多機能で高性能なマルチメディア機能と周辺ペリフェラルをSoCとして統合.
- MPEG-2/4のCODECは，720/480 30fpsまで処理でき，MPEG-4ビデオ再生時は20mW以下の低消費電力を実現.
- デュアルARM9プロセッサ（システム制御/コプロセッサ），ビデオ・プロセッサ，2Dグラフィック，LCDコントローラ，USB，IDE，AC97などの周辺I/Oコンポーネントが組み込まれている.
- LCDパネル，メモリ・カード，HDD，DVD-ROM，モデムなどの周辺デバイスを組み合わせることにより，ポータブル・ビジュアル・プレーヤ，マルチメディアPDA，DVD/DivXプレーヤ，ディジタル・カメラ，スマート・フォンなどを低コストと低消費電力で開発が可能.

●価格：
下記へ問い合わせ



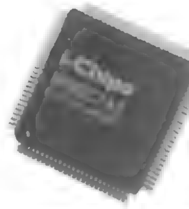
■共信テクノソニック（株）
TEL：03-5496-8805

●解像度変換LSI

IP00C742（SCL2）

- 液晶ディスプレイなどのドット・マトリクス型の表示デバイスに必要不可欠な，カラー・ディジタル画像の拡大を1チップで実行.
- NTSC～XGAまで広い範囲の画像入力に対応することができ，外付けのフレーム・メモリなしで，高速，高品位の画像拡大処理を安価に実現可能.
- ラスタ・スキャンのディジタル画像処理入出力と4線シリアルのCPUインターフェースを接続するだけで，画像拡大処理システムの構築が可能.
- 画像入力ポートは，RGB24ビット・ノンインターリーブ65MHz，YUV4：2：2 16ビットまたはYUV4：4：4 24ビット 65MHzをサポート.

●価格：下記へ問い合わせ



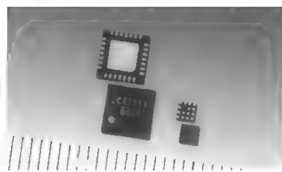
■アイチップス・テクノロジー（株）
TEL：06-6492-7277 FAX：06-6492-7388

●電源IC

LC41059GL/LC41059FN

- 小型，薄型で低ノイズを実現した，チャージ・ポンプ内蔵白色LED用の電源IC.
- チャージ・ポンプ回路を内蔵しているため，外付け部品は，コンデンサと抵抗のみ.
- 昇圧コイルを必要としないため，輻射ノイズがほとんど発生しない.
- チャージ・ポンプ回路による昇圧は，1/1.5/2倍から選択できるため，自由度の高い設計が可能となる.
- 最大2倍昇圧が行えるため，低い電源電圧でも比較的高い電圧を得ることが可能.
- 高精度カレント・ミラー回路の内蔵により，LED電流精度±0.5%以内で電流値を高精度に制御できるため，LEDの輝度ムラを抑えることができる.

●サンプル価格：¥200（LC41059GL）
¥220（LC41059FN）



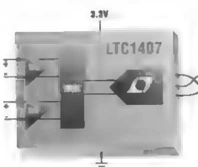
■三洋電機（株）
TEL：0276-61-8380 FAX：0276-61-9562

●A-Dコンバータ

LTC1407A/LTC1407

- LTC1407Aは同時サンプリング14ビット3Msps，LTC1407は12ビット3MspsのA-Dコンバータ.
- SO-8の半分の大きさの10ピンMSOPパッケージで供給されるため，コンパクトで高速データ収集システムの構築が可能.
- 2チャネルの差動入力をサポート.
- コマーシャル温度範囲とインダストリアル温度範囲を用意.
- LTC1407Aは，73.5dBの*SINAD*を達成し，ミッシング・コードのない14ビットを保証．単一3V電源で動作し，消費電力は標準12mW．無変換時の消費電力は，内蔵の2.5Vリファレンスがアクティブである NAPモードで3.3mW，すべてがパワー・ダウンするSLEEPモードで6μWに低減可能.

●サンプル価格：
LTC1407A ¥765～（1,000個時）
LTC1407 ¥440～（1,000個時）



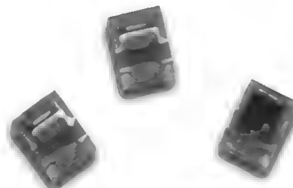
■リニアテクノロジー（株）
TEL：03-5226-7291 FAX：03-5226-0268

●照度センサ

HSDL-9001 照度センサ

- 携帯電話やPDAなどの携帯電子機器のバックライトを調節して，バッテリ寿命を伸ばすために用いられるセンサ.
- センサの出力は，検出された明るさに比例した直線性の高いアナログ出力となっている.
- コントロールICやホスト・プロセッサ側で，自由に照度検知レベルを設定することが可能.
- ピーク感度が550nm付近でかつ分光感度が人間の比視感度に非常に近い特性をもつ.
- 太陽光や蛍光灯，白熱灯やハロゲン灯など，多様な光源下でも同様の感度特性を維持できるため，正確な明るさの検出が可能.
- パッケージ寸法は0.6×2.0×1.5mmで，小型のQFNパッケージを採用.

●サンプル価格：¥80



■アジレント・テクノロジー（株）
TEL：0120-61-1280

●単体メモリ/小型メモリ・モジュール

DDR2 SDRAM SO-DIMM

- DDR2 SDRAMは，普及機向けの小容量DIMMに対応する，256Mビットの単体メモリ.
- SO-DIMMは，512ビットDDR2 SDRAMを搭載した，ノートPC向けの1Gバイト大容量のモジュール.
- サーバ/WS，デスクトップPCおよびノートPCとあらゆる情報処理システムに向けて，小容量から大容量品まで対応.
- DDR2 SDRAMは，0.11μmプロセスを適用し，667Mbpsの高速動作に対応予定.
- SO-DIMMには，DRAMを2段に積層する，独自のsFBGA技術を採用.

●価格：下記へ問い合わせ

■エルピーダメモリ（株）
TEL：03-3281-1648

●デュプレクサ/送信フィルタ

ACMD-7401 FBAR デュプレクサ/ACPF-7002 FBAR送信フィルタ

- ACMD-7401 FBARデュプレクサは，既存製品と比較して66%占有面積を削減することが可能で，セラミック製品と比較して，体積を約1/10程度に抑えることができる．送信帯域（1850～1910MHz）で1.8dB，受信帯域（1930～1990MHz）で2.2dBの低挿入損失のため，パワー・アンプなどで電流消費を低く抑えることが可能.
- ACPF-7002 FBAR送信フィルタは，同社のウェハでのチップ・スケール・パッケージング技術「マイクロキャップ」の採用により，占有面積を66%削減することが可能．米国PCS送信帯域（1850～1910MHz）に一つのフィルタで対応する，フルバンド送信フィルタを採用.

●サンプル価格：¥490（ACMD-7401）
¥110（ACPF-7002）

■アジレント・テクノロジー（株）
TEL：0120-61-1280

●UART

SC28L202

- 256バイトFIFOおよびリアルタイム・データ・エラー検出機能を内蔵した，低消費電力を実現する2チャネルUART.
- 動作電圧は－40～+85℃で，3Vおよび5V電源に対応.
- 通信，ネットワーキング，モバイル通信，コンピューティング，家電，生産管理，医療，セキュリティなどの用途に使用可能.
- 独自のリアルタイム・データ検出モードが装備されており，パリティ検査，巡回冗長検査，水平冗長検査などのタスクの負担からCPUを解放し，データの完全性を保証.
- 各FIFOのどのレベルでも割り込みが可能なため，割り込みエラーが最小限に抑えられ，CPUオーバ・ヘッドを大幅に低減.
- バス・サイクル時間は55nsで，完全な自動9ビット・モードXon/Xoffをサポート.

●価格：下記へ問い合わせ

■ロイヤルフィリップスエレクトロニクス
URL：http://jp.semiconductors.philips.com/

■弊誌では新製品に関するニュース・リリースを募集しております.
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：（03）5395-2127，E-mail：mngnews@cqpub.co.jp　（編集部）

●バス・スイッチ・ファミリ

B5S16861/B5S162861 B5S16862/B5S162862

- TTLパーツとピン互換のある高速CMOS 20ビット・バス・スイッチ4個を搭載したチップセット．
- ホット・プラグでのPCIカード挿入，電圧レベル変換，ノートPC用ドッキング・ステーション，メモリ・インタリーブ，汎用スイッチングなどのアプリケーション向けに設計．
- 電源電圧4V～5.5Vで動作し，入出力にはパワーダウン・プロテクションを持ち，改良されたラッチアップ耐性と2KVのESD保護機能を装備．
- 高オフ抵抗と低オン抵抗を備えており，オフ時にはアイソレーションを最大化し，オン時には負荷を最小化．

●サンプル価格：各¥90（1,000個時）



■STマイクロエレクトロニクス（株）
TEL：03-5783-8240 FAX：03-5783-8216

●オーディオ・アンプ

LM4667

- 携帯電話やコンシューマ向け，ポータブル・デバイス用のフィルタレス型Boomerオーディオ・アンプ．
- 単一電源式の完全集積，高効率スイッチング・オーディオ・アンプ．
- スイッチング・アンプとともに使われている出力フィルタを不要にするモジュレータを採用．
- デルタ-シグマ変調技術でアナログ入力処理を行うので，PWM方式と比較して，入力ノイズや前高調波ひずみ+ノイズの低減を可能にする．
- 3V単一電源で動作する場合，THD+N値が1%以下で，連続平均出力450mWを8Ωトランスデューサ負荷に与えることができる．
- 使用される電源電圧が多様に渡るため，2.7V～5.5Vまでの動作電圧範囲に対応．

●価格：¥70（1,000個時）

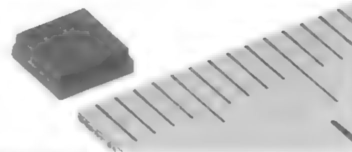
■ナショナル セミコンダクター ジャパン（株）
TEL：0120-666-116

●電力増幅器モジュール

BA01232

- W-CDMA方式の携帯電話端末向けにGaAsHBTを用いた，送信用電力増幅器モジュール（Hetero-junction，Bipolar，Transistor）．
- モジュール整合回路およびトランジスタ構造やサイズの最適化を行い，効率50%（出力電力26.5dBm時）と低アイドル電流35mA（動作電圧3.5V時）を実現．
- 電池小型化による端末の小型軽量化や通話時間向上が図れる．
- 金属キャップを樹脂封止に変え，内蔵チップの小型化や回路レイアウトの最適化により，容積0.02cc（4×4×1.2mm）にGaAs HBT増幅器（2段構成）と整合回路を搭載．

●サンプル価格：¥1,000



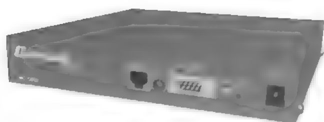
■三菱電機（株）
TEL：03-3218-4772 FAX：03-3218-4862
URL：http://www.MitsubishiElectric.co.jp/semiconductors/

●LANアダプタ

DWO-DIO8

- 接点などのディジタル入力，出力を，上位ホストとEthernetで接続し，接点の入出力の制御，状態監視が行える．
- NTPによる自動修正．
- 電源投入時，CPU内蔵のフラッシュ・メモリのチェックサム，DRAMおよびSRAMのRead/Writeの検査を行い，一つでもエラーが発生している場合，D_ERR LEDを点灯させ，プログラムを停止する．
- 接点入力を8点用意し，接点入力によりSNMPトラップを出力．
- 4台までのSNMPトラップ送信先の環境設定が可能．
- 接点出力を8点用意し，各接点のSNMPによる制御が可能．オン/オフは出力制御ごとに行う．

●価格：下記へ問い合わせ



■日本制禦機器（株）
TEL：072-661-4071 FAX：072-661-4065
E-mail：dwtec@nihon-seigyo.co.jp

●RS-232-C信号用モニタ・ボード

KIBシリーズ

- 機器間の長距離通信に使用する製品であり，シリアル通信を24時間連続して監視することが可能．
- 異常が発生した場合には，発生時にさかのぼって通信の解析を行うことができ，異常の特定による速やかな対策を可能としている．
- RS-232-Cなどのシリアル通信ラインの監視は，1枚のボードに2チャネル用意．
- シリアル入力コネクタは，Dサブ9ピンのオス．
- データの記録には，HDDを使用．
- 577M個のデータ記録保持が可能．
- 基本機能として「モニタ・ボードの管理」，「モニタ・ボードの操作」，「モニタ・データの表示」機能をもつ，コントローラ専用のソフトウェアを用意．
- ソフトウェアは，顧客の要望に特化した状態での提供，機能の追加も可能となっている．

●サンプル価格：¥500,000

■ケル（株）
TEL：042-374-5801 FAX：042-374-5887

●シーケンサ・ミラーリング・ユニット

MELCOM-01

- シーケンサどうしを無線通信で結び，双方のデータをつねに同一の状態に保つ，プログラムレスな特定小電力双方向無線機．
- 一般的な汎用モデム型無線機と異なり，内部にシーケンサ通信専用の演算機構を組み込むことにより，機能を特化し，高速応答性を実現．
- シーケンサの通信規格の一つである「専用プロトコル型式4」に対応しているため，無線機との通信プログラミングが不要．
- 正常に通信していることを示す信号出力機能を搭載することで，データが最新であるかどうかの判定が可能．
- 送受信データ量を必要に応じて，2,4,6,8バイトから選択可能．
- 子機から親機に向け，一方通行でデータを送る単方向モード機能を搭載．
- 親機に高感度アンテナなどを装着することで，通信距離1km以上も可能．

●価格：下記へ問い合わせ

■大成ラミック（株）
TEL：0480-97-0194 FAX：0480-97-0910

●シリアル-ファイバ・メディア・コンバータ

TCF-142シリーズ

- 複数のインターフェース回路が搭載されているため，RS-232-CまたはRS-422/485シリアル・インターフェースと，マルチまたはシングルモードの光ファイバ間の変換処理が可能．
- 2km（TCF-142-M：マルチモード・ファイバ），または20km（TCF-142-S：シングル・モード・ファイバ）までのシリアル転送距離を延長することが可能．
- 電源ケーブルを誤った端末に接続しないように，特別な保護を提供する逆電源保護機能をサポート．
- コンバータは，電源ワイヤのプラス，マイナスを自動的に検出し，その結果に従って電源装置を調整するように設計されている．
- レジスタの強度を無効にする，変更するなどの設定は，筐体外部のディップ・スイッチで行える．
- ハードウェア的にシリアル信号のボーレートを自動的に検出するため，装置のボーレートが変化しても，信号は問題なく伝送される．

●価格：下記へ問い合わせ

■MOXAテクノロジーズ社
TEL：+886-2-8919-1230 FAX：+886-2-8919-1231

●ワンボード・コンピュータ

A6版CPUボード

- コンパクトなA6サイズのため，小さなスペースへの組み込みが可能．
- CPUには，BGAタイプ（Celeron 400MHzまたは低電圧版Pentium III 933MHz）を搭載可能．
- チップセットは，Intel 815Eを使用．
- 拡張PCI，PCMCIAスロット，コンパクト・フラッシュ・ソケット，LAN，USB，シリアル・ポート，サウンド，パラレル・ポート，NTSC入力（キャプチャ機能付き），および画面出力用としてTV出力，アナログRGBコネクタ，DVIを搭載．
- 電源は，ACアダプタからの12～15Vで動作．
- コンパクト・フラッシュ内にWindows XP/2000，LinuxをROM化して動作させることができる．
- 消費電力は，1.2A（Celeron 400MHz起動時）．

●価格：下記へお問い合わせ

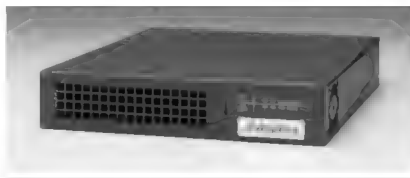
■（株）アイ・シー・エー
TEL：0480-26-1566
URL：http://www.ica-ua.co.jp/

●IP電話装置

Applico SIP RTC スイッチ ASA280

- インターネットでVoIPを利用する際に，通信品質とセキュリティの問題を解決する．
- マルチ・ストリーム技術により，画像，アプリケーション，プレゼンスなどのRTC（リアルタイム・コミュニケーション）に対応．
- n:nのNATを越え，VPN，PPTP，TLSなどの暗号化技術に対応．
- 最大ユーザ登録1000，最大同時セッション400のキャパシティを持つ．
- G711，G723，G723.1，G729などの，さまざまなCODECに対応．
- Microsoft Windows Messengerとの互換性を持つ．

●予定価格：¥980,000～



■（株）アズジェント
TEL：03-5643-2561 FAX：03-5643-2571
E-mail：info@asgent.co.jp
URL：http://www.asgent.co.jp/

●ICカード・ソリューション

ASECard for Windows

- ICカードを利用して，Windows Server 2003/2000 ServerのPKIをベースとしたセキュリティ機能を活用し，低コストでネットワーク・セキュリティを実現するソリューション．
- ICカードを利用した2因子によるユーザ認証に加え，RSA公開鍵暗号技術を併用し，セキュリティ・レベルの強化を図っている．
- 必要なアプリケーションは，すべてWindowsサーバに標準搭載されているため，追加のソフトウェアは不要．
- 導入キットは，ICカード，ICカード・リーダ（ASEDriver III e），暗号ミドルウェア（ユーザ数無制限ライセンス付き），管理用ユーティリティで構成される．

●価格：¥9,800（導入キット）

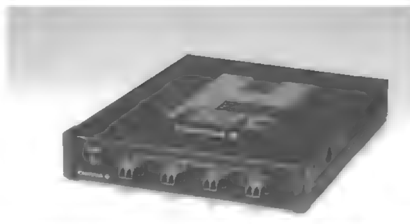
■（株）アテナ・スマートカード・ソリューションズ
TEL：0426-60-7555 FAX：0426-60-7106
E-mail：ase@athena-scs.co.jp

●光パケット・スイッチ

40Gbit/s 光パケットスイッチ

- 光パケット信号を電気信号に変換することなく，光のまま高速で切り替える「光スイッチ素子」と，光パケット信号の宛て先情報に基づいて光スイッチを制御する「光電子融合回路」の組み合わせで構成．
- 「光スイッチ素子」は，化合物半導体に流す電流をオン/オフすることで光の経路を切り替えることができ，2ns以下という高速での光パケット信号切り替えを実現．
- 「光電子融合回路」は，光パケット信号の先頭部分に記録された宛て先情報を読み取る高速受光素子と，その情報を処理する高速電子回路を組み合わせたもので，認識した宛て先情報で光スイッチ素子を制御する．

●価格：下記へ問い合わせ



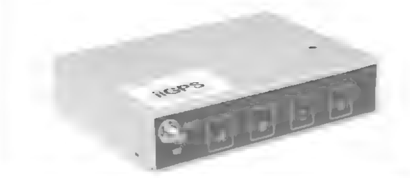
■横河電機（株）
TEL：0422-52-5530 FAX：0422-55-6492

●ドライブ・レコーダ

PVTレコーダ

- 高精度GPSとGPS制御技術を用いて，Position（緯度，経度，高度），Velocity（速度），Timing（時刻）を毎秒測位し，車両の位置，速度，時刻，進行方向をデータとして保存できる．
- 誤差は，DGPSなどの補正を使用せずに，P：2m以内，V：0.2km/h以内，T：95ns以内．
- Type IIのCFスロットを持ち，PHS/DoPa/FOMA/無線LANなどの通信カードや，CFメモリ・カード，マイクロドライブなどの記録媒体を装着することが可能．
- リアルタイムでの車両位置確認のみならず，メモリ・カードを介したオフラインの運行記録システムも可能．

●サンプル価格：¥50,000



■（株）ジーシーシー
TEL：03-5209-6355

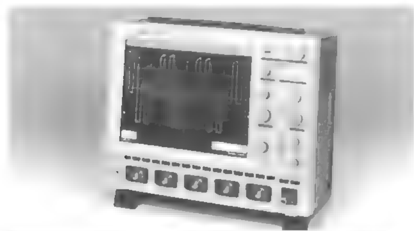
■弊誌では新製品に関するニュース・リリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：（03）5395-2127，E-mail：mngnews@cqpub.co.jp （編集部）

●ディジタル・オシロスコープ

サーファー5シリーズ

- Windows XPを採用し，mathcadやMatlabなどの市販ソフトウェアを使用することによって，解析を1台で行うことが可能．
- 従来機種と比較して，厚さは約半分の152mmを実現．
- 10.4インチの高輝度，SVGAカラー液晶を採用することで，さまざまなパラメータや解析データを波形と同時に表示することが可能．
- タッチ・スクリーンによる操作が可能．
- 上40度，下70度，合わせて110度，左右70度，合わせて140度の広視野角を実現．
- メモリ長やサンプリング速度で用途に合わせたラインナップを用意．

●価格：¥498,000～¥998,000



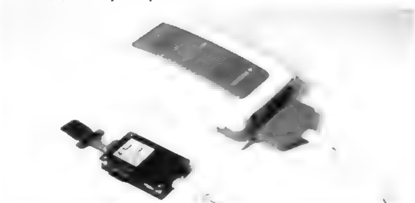
■岩通計測（株）
TEL：03-5370-5474 FAX：03-5370-5492
E-mail：info-tme@iwatsu.co.jp

●SDカード・プロトコル・アナライザ

AX200

- SDメモリ・カード専用のプロトコル・アナライザ．
- SDカード対応機器とSDカード間で交換されるコマンド，レスポンスおよびデータをタイム・スタンプ付きで表示するため，プロトコル解析と同時に転送性能の評価，解析などを行うことができる．
- 特定コマンド・パターンやエラー発生などの条件を指定しておくことで，探している現象を素早く探知可能．
- 横河電機製のディジタル・オシロスコープと組み合わせることで，プロトコルだけではなく，そのときの実際の信号波形も連動させて解析することができるため，効率的なハードウェア検証が可能．

●価格：¥1,300,000



■横河電機（株）
TEL：0120-137046 FAX：0422-52-6624

●無線LAN用IPコア

Jennic 2.4GHz IEEE802.15.4 IP

- 0.18μm RFCMOSでシリコン実証済みの2.4GHz IEEE802.15.4準拠の無線コア．O-QPSKモデム，ベースバンド・コントローラ，およびMACプロトコル・スタックで構成．
- 無線コアは，2.4～2.5GHz ISM周波数帯で動作し，シングル・エンドで抵抗を考慮した，差分型RFポートをもち，アンテナとのインターフェースを取るための外部コンポーネントが不要．
- モデムは，ベースバンド信号のO-QPSKスペクトル拡散型変調/復調を行い，4ビットのベースバンド信号を16個の疑似直交32ビット・コードに250kbpsで変換し，2Mchips/sのオンエアー・チップレートを実現．
- ベースバンド・コントローラは，スーパ・フレームとプロトコル・タイマ，リトライ付き自動認識，およびCSMA/CAアクセスなど，下層のMAC機能の多くを回路に実装している．

●価格：下記へ問い合わせ

■ジェニック社
TEL：+44(0)114 281 2655 FAX：+44(0)114 281 2951
E-mail：info@jennic.com
URL：http://www.jennic.com/

●CFカード・タイプ・アナログ入出力

AXC-AC01 AXC-AD01 AXC-DA01

- AXC-AC01はアナログ入出力，ディジタル入出力機能を，AXC-AD01はアナログ入力とディジタル入出力，AXC-DA01はアナログ出力とディジタル入出力機能をサポート．
- コンパクト・フラッシュ仕様R1.4準拠のCF+Type Iカードで提供．
- 24.5MIPSのMPUを内蔵したことにより，高速なA-D，D-A変換処理が可能．
- 2チャネル16ビット・アナログ入力機能は，0～2.45Vの入力信号を，最大1Mspsの高速サンプリングで16Kワードのメモリに格納することが可能．
- 12ビット・アナログ出力機能は，10μsのセトリング・タイムにて0～2.43Vの電圧を出力．
- A-D変換トリガは，タイマによる一定周期A-Dサンプリングや外部トリガによるA-Dサンプリングなどのサンプリング機能を持つ．
- 2チャネル(シングルエンド入力)または1チャネル(疑似差動入力)の入力が可能．

●価格：AXC-AC01 ¥62,000/AXC-AD01 ¥57,000/AXC-DA01 ¥54,000

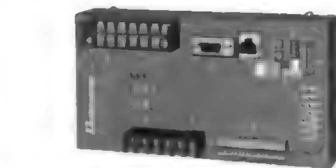
■（株）アドテックシステムサイエンス
TEL：045-331-7575 FAX：045-331-7770
E-mail：sales@adtek.co.jp

●産業用リモートI/Oソリューション

Dataway Terminal Systemシリーズ

- データ収集，分散制御，PLCレス化などの用途に適する．
- 各ユニットにネジアップ式端子台を採用しており，配線作業の簡素化を実現．
- 入出力部に端子台を採用することで，外部配線をダイレクトに接続可能．
- 電源，通信ラインに2ピース端子台を採用することで，システムを止めずにユニットの交換が可能．
- マスタ・ユニットを2台置くことにより，マスタの完全2重化を実現．
- スレーブ間の通信をRS-485で行っているため，分散制御が可能．
- スレーブ・ユニットは，パソコンやPLCのリモートI/Oとして単体で利用することが可能．

●価格：下記へ問い合わせ



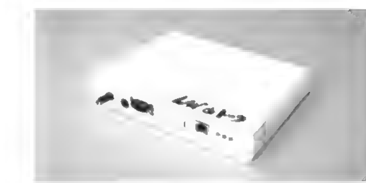
■日本制禦機器（株）
TEL：072-661-4071 FAX：072-661-4065

●インスタント・ボイス・コミュニケーション・システム

LAN de トーク

- パソコンを介することなく，Ethernetコネクタに差し込むだけで，通話が可能．
- イヤホン・マイクにより，ハンズフリーでの会話が可能．
- 16台同時通話が可能となっており，各種電気工事のツールとして利用可能．
- 一斉呼び出しのスイッチ付き．
- 通常/通話チャネルの切り替えが可能．
- 通話チャネルごとに，個別通話が可能．
- 外部スピーカ端子に接続可能．
- 小型，省スペース設計．

●価格：オープン



■（株）ハウ
TEL：042-753-3616 FAX：042-769-7300
E-mail：info@how.jp
URL：http://www.how.jp/

●開発ツール

インテル・スレッド・チェッカー2.0

- 競合状態，ストール，デッドロックなど，ソフトウェアのスレッディングの問題を検出．
- スレッディングのバグを自動的に検出して抽出する，エラー検出エンジンを使用．
- 既存のデバッグ技術では検出が難しい，再現不可能なバグの検出が可能．
- バグの原因となるソース・コードの問題箇所，参照したメモリを検出．
- Win32 APIのスレッド，Cランタイム・ライブラリ関数，OpenMPと互換性がある．
- VTuneパフォーマンス・アナライザ7.0またはそれ以上へのアドオン・ツール．
- スレッド・プロファイラを利用すると，Win32およびOpenMPスレッド化モデルによるマルチスレッド・アプリケーションのパフォーマンスを解析し，ボトルネックを特定できる．

●価格：¥156,000（VTune7.1付き）
¥91,000

■エクセルソフト（株）
TEL：03-5440-7875 FAX：03-5440-7876
URL：http://www.xlsoft.com/intel/
E-mail：intel@xlsoft.com

●開発支援ツール

DataObjects for .NET/ WebDataObject for .NET/ Menues&Toolbars for .NET

- DataObjects for .NETは，データベースとデータ連結コンポーネントの橋渡しをコーディングなしで行えるツール．データベース構造をデータ・スキーマに変換し，ビジネス・ロジックをグラフィカルに取り込んで，データセットの生成やデータの更新を行う．分散3階層アプリケーション開発の自動化を可能にする．
- WebDataObject for .NETは，Webフォームで DataObjects for .NETの機能をすべてサポートするほか，ASP.NETに特化した機能を搭載．アクセスが集中するデータをメモリに置く「サーバサイド・キャッシング」機能をプロパティ設定だけで実現．
- Menues&Toolbars for .NETは，メニュー，ツールバー，Outlookバーを Windowsアプリケーションに実装する，フォーム用のコンポーネント製品．

●価格：DataObjects for .NET ¥78,000
WebDataObject for .NET ¥78,000
Menues&Toolbars for .NET ¥68,000

■グレープシティ（株）
TEL：022-777-8211 FAX：022-777-8233
E-mail：sales@grapecity.com

●メッシュ・モーフィング・ツール

ANSYS ParaMesh

- 既存の有限要素モデル，あるいは過去の有限要素モデルを利用可能．
- 飛行機や自動車車体など，何百万自由度をもつ大規模モデルの読み込み，変更が可能．
- CADライセンスや専門家がいなくても，解析モデル形状の変更が可能．
- 既存のメッシュを利用することで，設計形状のパラメータ・スタディにはメッシュ再分割が不要．
- 設計変更に対応し，新規メッシュの生成が可能．
- CADモデルなしで，形状の最適化が可能．
- 精度の高いDOEが可能．
- マルチフィジックス・アプローチが可能．
- メッシュは，構造解析，固有値解析，流体力学，電磁場，音響解析などの複数物理分野の対応が可能．
- 自動メッシュ・モーフィング機能で，新規メッシュ作成時間を大幅に短縮．
- 複数の形状パラメータに対して，同時にメッシュ・モーフィングが可能．

●価格：¥2,000,000

■サイバネットシステム（株）
TEL：03-5978-5406 FAX：03-5978-5960
E-mail：anssales@cybernet.co.jp

●認証サーバ・ソフトウェア

RSA ACE/Server 5.2

- レポート作成やデータベースとの同期化など，運用管理機能を強化した，SecurIDからの認証要求を処理する認証サーバ・ソフトウェア．
- 定義済みクエリやSQLで自由に検索条件を作り，レポートの作成が可能．
- レポートはスケジュール機能により自動的に生成され，これによってシステムの運用状況の把握が可能．
- LDAPとRSA ACE/Serverユーザ・データ・ベースの同期項目の細かな設定が可能．同期はスケジュール機能により自動実行できるため，マニュアルによる変更つどの同期作業が不要となるため，管理コストの低減を実現．
- すべてのサーバのイベント・ログの記録が可能で，高度な監査要求への対応が可能．

●価格：¥719,000～
（25ユーザ/ベース・ライセンス）
¥971,000～
（25ユーザ/アドバンスド・ライセンス）

■RSAセキュリティ（株）
TEL：03-5222-5230
E-mail：info-j@rsasecurity.com
URL：http://www.rsasecurity.co.jp/

●.NET対応開発ツール

DevPartner Studio Professional Edition7.1 英語版

- リアルタイムにアプリケーションのメモリ使用状況の確認が可能．
- RAMフットプリント，テンポラリ・オブジェクトの発生状況，メモリ・リークなどの情報をグラフィカルに表示．
- コール・グラフ表示機能と連動し，メモリを使用しているメソッドの表示，およびほかのメソッドとの呼び出しの相関関係の表示が可能．
- メモリ使用の非効率なメソッドを迅速に特定して最適化し，パフォーマンスおよびメモリ使用率の改善が可能．
- 実行時間の測定によるボトルネック検出機能に加え，コード修正前と修正後のパフォーマンスの比較を表示．
- コール・グラフ表示機能により，メソッド呼び出しのフローを追いながらパフォーマンスのボトルネックとなるコードを素早く突き止め，比較機能を使用してコード修正効果を定量的に判定することが可能．

●価格：¥328,000

■日本コンピュウェア（株）
TEL：03-5473-4530 FAX：03-5473-4528
E-mail：marketingjapan@compuware.com
URL：http://www.compuware.co.jp/

●開発ツール

Borland Delphi 8 for the Microsoft .NET Framework 日本語版

- .NET Frameworkの仕様に準拠した開発をサポートしており，信頼性，セキュリティ，相互運用性の高い.NETアプリケーションの開発が可能．
- データベース・アクセスを含むコンポーネントによるビジュアル開発をサポートするVCLを.NET Framework向けに対応させたVCL for .NETを実装．
- 過去のアプリケーション資産を.NETアプリケーションとして移行し，再利用ならびに機能拡張が可能．
- .NET Frameworkにおける強力なWebアプリケーション開発フレームワークであるASP.NETに対応しており，XML Webサービスや多機能なWebアプリケーションの開発が可能．
- 言語コンパイラ，デバッガ，CodeInsightなどの支援機能が搭載されたエディタ，Windows Forms，VCL Forms，Web Formsなどのデザイナやオブジェクトインスペクタなどの LiveToolが統合されたRADスタイルの統合開発環境．

●価格：¥68,000～¥432,000

■ボーランド（株）
TEL：03-5323-3071 FAX：03-5323-3072

■弊誌では新製品に関するニュース・リリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp （編集部）

IPパケットの隙間から

強引なセールスとネット社会の成熟

祐安 重夫

1人で仕事をしていて迷惑なのは，こちらのつごうもおかまいなしの電話やセールス・マンである．昼すぎから翌朝にかけてが筆者の仕事時間なので，午前中の訪問者や電話は最初から無視している．さすがに留守番電話にメッセージを残していくようなセールス電話はないようで，半覚醒状態で聞いていると，留守電に切り替わったとたんに電話を切ってしまうパターンが多い．メッセージを残そうという気配がある場合は，さすがに寝床から起き出して電話に出る．

もっとも筆者の生活習慣は，友人知人や仕事の関係者ならだいたいは知っているので，わざわざ午前中に電話をかけてきたりしない．逆に夜遅くや深夜に，仕事の打ち合わせを電話でする場合はある．まあ，たいていの用件はメールで片付くし，資料の送付などを含めてそのほうが便利であることもたしかである．

資料の送付についていえば，FAXも月に1回のこの原稿の校正をのぞけば，ほぼ100%がダイレクト・メールである．

不思議なことに，午後にかかってくるセールス電話というのはあまりなく，夕方の5時をすぎると突然始まる．

たいていは「社長」宛てにかかってくるので，「社長はいない」，「帰ってこない」，「明日もいない」で通すし，ときには「2度と電話をしないように」とまでいいきる．仕事の電話なら，「社長」宛てにかかってくるはずはない．「社長」本人が自分はいないといっているのだから，これほど確かなことはない．

セールスについては，5年以上前に，午後から夕方にかけて同じ新聞の勧誘員が，4時間ほどの間にとっかえひっかえ7人もやってきたことがあった（たぶん午前中にも来ていたのだろうが，もちろん眠っていたので気がつかなかった）．このときに腹を立てて，仕事場の玄関先に「セールスや勧誘を目的とする訪問者がインターホンのボタンを押したら悪質な業務妨害として警察に通報する」という紙を貼ったら，それからはほとんど来なくなった．

たまにやってくる輩（やから）は，「お前は字が読めないのか」といって追い返すが，中にはわざわざ厭味（いやみ）でインターホンのボタンを押すセールス・マンもいる．そういうときはこちらもドアの横に置いてある大型のスパナを手に持って，わざとドアの隙間から見せたりしているので，あまり人がいいとはいえないが．

ときどき，大手企業のセールス・マンがやってきて，こちらが文句をいうとひたすら「すいません」と謝って帰っていく場合があるが，大手企業なら，それなりのセールス・マン教育をすべきではないだろうか．

しかし，この間やってきたのは，ちょっとひどかった．会社の名前はXXといって，インターホンにでたら突然，「光ファイバを引きにうかがいました」ときた．もちろん，そんなものは頼んでいない．こういう表現でセールスに来るケースは，その昔，現在のNTT以外の電話会社ができたときに，その代理店が電話会社を自動選択するアダプタを強引に付けにくるというパターンと似ている．そのときもあまりに強引で，無料なのだから付けて当然という態度に腹が立って即座に追い返し，置いていった名刺の番号に電話して，上司にあの態度はなんだと抗議した．

今回のXXは，直接聞いた話とインターネットから得た情報を総合すると，光ファイバとIP電話とインターネット接続を組み合わせたサービスを提供するということのようだが，電話番号が050からのものにならず，今までのままであるという点を売り物にしているようだ．しかし，どうやら専用の電話機もセットになっているという点で，2002年の12月号の本連載で書いた，母の会社にアナログ電話機がもうすぐ使用できなくなると嘘をついて，高価なディジタル電話機を売り付けようとした悪徳商法を思い出してしまった．XXの商売も，あからさまな悪徳商法の臭いがする．

ところで，XXという会社名に記憶があったので，Googleで「XX 悪徳商法」を検索してみたら，少なからぬ情報が出てきた．その記憶にはまちがいはなく，悪徳商法でずっと以前にも話題になったことがある会社だということがわかった．やはりGoogleは便利である．

と思っていたら，悪徳商法マニアックスのWebサイト，

http://www6.big.or.jp/~beyond/akutoku/

で悪徳商法と名指しされていた会社がGoogleにクレームをつけて，Googleの検索結果の中からこの会社のことを書いた悪徳商法マニアックスのページを表示されないようにさせたらしい．

もっとも，このことはすぐ公になり，この問題を取り上げたWebページが大量に発生したので，クレームをつけてきた会社にとっては逆効果になったかもしれない．またこれに関連して，Google以外にもクレームをつけられたサイトがあり，そのクレームを受け入れてしまったために，評判を落としたところもあるようだ．

Googleはその影響力を考えると，ネットワークの外の世界におけるジャーナリズムと同等の成熟度が求められるのではないだろうかということを，この件でつくづくと感じさせてくれた．もっとも，最近は外の世界でも，あまり成熟しているとはいえないジャーナリズムが多くなってはいるのだが．

すけやす・しげお　インターメディアアクセス



INFORMATION
海外・国内イベント/セミナー情報



海外イベント

3/29-4/2 **electronica USA with the Embedded Systems Conference**
Moscone Convention Center, San Francisco, CA, USA
CMP Media
http://www.esconline.com/

4/5-7 **DesignCon East 2004**
Holiday Inn Boxborough Woods, Boxborough, MA, USA
International Engineering Consortium
http://www.iec.org/events/2004/designcon_east/

4/14-17 **Hong Kong Electronics Fair 2004（Spring Edition）**
Hong Kong Convention and Exhibition Centre,
Wanchai, Hong Kong, China
Hong Kong Trade Development Council
http://hkelectronicsfairse.com/

4/17-22 **NAB2004**
Las Vegas Convention Center, Las Vegags, NV, USA
National Association of Broadcasters
http://biz.knt.co.jp/nab/

4/19-24 **Hannover Messe 2004**
Hannover Messe, Hannover, Germany
Deutsche Messe AG
http://www.hannovermesse.co.jp/

5/4-6 **Assembly East**
Hynes Convention Center, Boston, MA, USA
Reed Exhibitions
http://www.assemblyeast.com/

5/9-14 **NetWorld + Interop LAS VEGAS 2004**
Las Vegas Convention Center, Las Vegas, NV, USA
MediaLive International
http://www.interop.com/lasvegas2004/

国内イベント

4/4-8 Intel Developer Forum Japan 2004
ヒルトン東京ベイ（千葉県浦安市舞浜）
インテル
http://www.intel.co.jp/jp/idf/

4/7-8 IP FORUM 2004
東京国際展示場（東京ビッグサイト，東京都江東区）
リックテレコム
http://www3.ric.co.jp/expo/ip2004/

4/7-9 EDEX2004 電子ディスプレイ展
東京国際展示場（東京ビッグサイト，東京都江東区）
電子情報技術産業協会
http://edex.jesa.or.jp/

4/7-9 センサ総合展 2004
東京国際展示場（東京ビッグサイト，東京都江東区）
日本工業新聞社
http://www.jij.co.jp/event/sensor/index.html

4/13 第3回 UML ロボットコンテスト
青山テピア（東京都港区）
オブジェクトテクノロジー研究所（OMG日本代表）
http://www.otij.org/umlforum2004/robocon/

4/14-16 2004 マイクロエレクトロニクスショー
最先端実装技術・パッケージング展
東京流通センター（TRC，東京都大田区）
エレクトロニクス実装学会
http://www.jiep.or.jp/meshow/index.html

4/20-21 Wi-Fi Planet Conferences & Expo Japan 2004
新宿 NSビル NSイベントホール（東京都新宿区）
IDGジャパン，Jupitermedia
http://www.idg.co.jp/expo/wi-fi/

4/21-22 パワーエキスポ 2004
東京国際フォーラム（東京都千代田区）
パワー・エキスポ実行委員会
http://www.a-tex.co.jp/power-expo/

開催日，イベント名，開催地，問い合わせ先の順
日程はすべて予定です．問い合わせ先にご確認のうえ，お出かけください．

セミナー情報

SH-Linux マイコン入門
開催日時　：4月1日（木）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎ 03）5395-2125，FAX（03）5395-1255

Linux デバイス・ドライバ入門
開催日時　：4月2日（金）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎ 03）5395-2125，FAX（03）5395-1255

初めての Visual Basic 6.0 徹底マスター
開催日時　：4月6日（火）～4月7日（水）
開催場所　：ディーアイエステクノサービス研修室（東京都文京区）
受講料　：87,000円（税込）
問い合わせ先：（株）エイチアイ ICP事業部，☎ 03）3719-8155，FAX（03）5773-8661
http://icp.hicorp.co.jp/seminar/VB/vb_1.asp

C言語ポインタ徹底習得（ポインタを正しく教える方法）
開催日時　：4月8日（木）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎ 03）5395-2125，FAX（03）5395-1255

シミュレータ テクニカルトレーニング
開催日時　：4月9日（金）
開催場所　：ガイオ・テクノロジー日本橋事業所セミナールーム（東京都中央区）
受講料　：無料
問い合わせ先：ガイオ・テクノロジー（株），E-mail：seminar@gaio.co.jp
http://www.gaio.co.jp/event/regular_seminar.html

リチウムイオン2次電池と充電回路の基礎
開催日時　：4月9日（金）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎ 03）5395-2125，FAX（03）5395-1255

無線データ通信の基礎と2.4GHz帯無線LAN
開催日時　：4月15日（木）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎ 03）5395-2125，FAX（03）5395-1255

～新人，新任者のための～オブジェクト指向入門技術解説
開催日時　：4月15日（木）～4月16日（金）
開催場所　：SRCセミナー・ルーム（東京都高田馬場）
受講料　：73,000円（税別）
問い合わせ先：（株）ソフト・リサーチ・センター，☎ 03）5272-6071，FAX（03）5272-6345
http://www.src-j.com/teiki_no/Src/new_it_2.htm

スミス・チャートを使った高周波回路設計の基礎
開催日時　：4月16日（金）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎ 03）5395-2125，FAX（03）5395-1255

シミュレータファミリ入門体験コース
開催日時　：4月16日（金）
開催場所　：ガイオ・テクノロジー日本橋事業所セミナールーム（東京都中央区）
受講料　：無料
問い合わせ先：ガイオ・テクノロジー（株），E-mail：seminar@gaio.co.jp
http://www.gaio.co.jp/event/regular_seminar.html

ディジタル信号処理入門
開催日時　：4月17日（土）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎ 03）5395-2125，FAX（03）5395-1255

USB2.0ターゲット・システムの設計事例
開催日時　：4月22日（木）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎ 03）5395-2125，FAX（03）5395-1255

ソフトモジュールテスト用シミュレータ WinAMS 体験コース
開催日時　：4月23日（金）
開催場所　：ガイオ・テクノロジー日本橋事業所セミナールーム（東京都中央区）
受講料　：無料
問い合わせ先：ガイオ・テクノロジー（株），E-mail：seminar@gaio.co.jp
http://www.gaio.co.jp/event/regular_seminar.html

DSPによるディジタル・フィルタ入門
開催日時　：4月24日（土）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎ 03）5395-2125，FAX（03）5395-1255

PalmOS プログラミング基礎
開催日時　：4月27日（火）～4月28日（水）
開催場所　：ディーアイエステクノサービス研修室（東京都文京区）
受講料　：80,000円（税込）
問い合わせ先：（株）エイチアイ ICP事業部，☎ 03）3719-8155，FAX（03）5773-8661
http://icp.hicorp.co.jp/seminar/palm/palm.asp

読者の広場

Interfaceへの声



2004年3月号特集「Cプログラミングの基礎知識」に関して

▷間違いやすいコーディング例は，基礎知識を再確認できて非常によかった．デバッグや組み込みに関する記事は，もっと深く掘り下げて欲しかった．次回に期待します．
(moto)

▷中島氏の記事はDOS時代よりいろいろ読ませていただいております．C言語で測定機器や計測機器開発にたいへん役に立ちました．書棚にある中島氏の書籍は私の宝物です．Windows時代またC++の世界になってきました．I/OボードやA-D/D-A変換ボード関連も，VBとOCXで十分に実用的なものが実現できてしまいますが，C言語を使った高速な処理のテクニックなどの書籍を，Windows版でお願いします！
(ヤマモト)

▷しばらくぶりにC言語を思い出させてくれるには十分な内容でした．意外と理解せずにプログラムを組んでいたということに気づかされた．また組み込みCプログラミングも今後のためになりそうでよかった．
(KAZZU)

アンケートの結果

興味のあった記事（2004年3月号で実施）

①第1章　Cで間違いやすいコーディング例
②第2章　コーディングの違いと最適化例
③第3章　関数作成の勘所
④第4章　デバッグの前準備と心得
⑤Prologue　これからCプログラミングを始める人へ
⑥第5章　組み込みCプログラミング
⑦初級ドライバ開発者のためのWindowsデバイスドライバ開発テクニック（第6回，最終回）
⑧やり直しのための信号数学（第21回）
⑨開発環境探訪（第25回，最終回）
⑩シニアエンジニアの技術草子（参拾六之段）
⑪IPパケットの隙間から（第60回）
⑫移り気な情報工学（第37回）
⑬TOPPERSで学ぶRTOS技術（第5回）
⑭CQ RISC評価キット/SH-4PCI with Linux活用研究5
⑮TMS320C6713搭載DSPスタータキットを使ったC++によるDSPオブジェクト指向プログラミング（第2回）
⑯開発技術者のためのアセンブラ入門(第23回)
⑰ハッカーの常識的見聞録（第39回）
⑱PowerPC G4の概要とAltiVecを活かしたプログラミング技法
⑲「VxWORKS」を使ったRTOS技術の基礎と応用（第4回）
⑳TRONSHOW 2004
㉑Engineering Life in Silicon Valley



特集担当デスクから

☆「あなたはどのような仕事をしていますか」，「組み込み技術者をやっております」，「その『組み込み』ってなんですか？」，「えーと…」．
☆弊誌でたびたび使われるキーワード「組み込み」ですが，このことばの指す意味がまったくわからないという読者は少ないと思います．しかし，普段何気なく使っているこのことばも，組み込みに縁のない人に説明するためには多言を要します．ことばが説明しにくいということは，仕事内容の説明もしにくいということを意味します．「組み込みってどんな仕事なんだろう？」そんな素朴な疑問に答えるために，今回の特集を企画しました．
☆組み込みはおもしろい仕事です．コスト，サイズ，消費電力…あらゆる制約の下で高品質な製品を追求する，そんな世界が待っています．世間では泥臭いといわれるような技術から，システム全体を設計するための大局的な技術まで，総合的な力が問われる分野です．巨大システムの一部ではなく，製品のすべてを数人で開発することも少なくありません．そんな「やりがい」のある仕事です．
☆実際に現場に入ると，「自分はソフト屋/ハード屋だから…」，「この言語は使ったことがないから…」，「このチップは初めてで…」，「この納期/コストでは…」などのように，さまざまな困難にぶつかるかもしれません．しかしそれらを一つずつ解決し，製品を作り上げる喜びはほかの分野では味わえないかもしれません．
☆にも関わらず，これまで組み込みに脚光が当たることはあまりありませんでした．あくまでも産業を支える縁の下の力持ち，そんな存在でした．しかし近年の組み込みブーム（？）により，この分野に注目が集まりつつあります．多くの若い人がこの分野を目指し，活気を与えてくれるに違いありません．
☆また，問題解決のためには，自分の専門分野だけでなく，ほかの分野に対する知識も必要になることがあります．ソフトの書けるハード技術者，回路図の読めるソフト技術者が理想的です．そんな「総合力」をつけるためにも，自分とは直接縁のない分野にも興味を示すようにすると良いかもしれません．やはり一生勉強，なのでしょうか．



特集『Cプログラミングの基礎知識』についてのアンケートの結果

Q1　CPU内蔵組み込み機器のプログラミングをしたことがありますか？

①はい(87%)　②いいえ(13%)

Q2　使用したプログラミング言語は何ですか？(複数回答可)

①C++(14%)
②C(34%)
③アセンブラ(29%)
④Java(3%)
⑤BASIC(11%)
⑥Perl(3%)
⑦Pascal(6%)
⑧その他(0%)

Q3　現在興味のあるものは何ですか？

①PC関連情報(24%)
②CPU関連情報(26%)
③ストレージ関連情報(13%)
④ロボット関連情報(26%)
⑤その他(19%)
開発品質，開発言語と環境，ネットワーク関連，無線応用ほか

Q4　今後本誌で取り上げて欲しい記事があれば，教えてください．

ロボット関連，組み込み分野のセキュリティ，Linux2.6のリアルタイム性，μITRON関連，テスト方法ほか

Interface編集部からのお知らせ

プロ技術者のための ディジタル信号処理技術研究会への誘い

現在あらゆる分野において，ディジタル信号処理技術が使用されています．

アナログ信号を取り込んだ後に行う信号処理の多くが，じつはディジタルによる信号処理で行えることは昔からわかっていました．そして，半導体技術の発展と相まって非常に幅広く利用されるようになってきました．しかし，この分野はまだ混沌としており，まだまだ発展途上にあるようです．フィルタなどによる処理，統計処理から，オーディオ/ビデオ信号の圧縮/伸張，3次元データ処理，変調/復調など，多角的ではあるけども共通要素をもった技術発展があるように感じられます．ハードウェアもDSP専用のもの，すでに汎用CPUのライブラリになってしまったもの，FPGA用のIPになってしまったものなどさまざまです．

このようなディジタル信号処理に携わっている技術者の方は相当に多いと考えています．しかしながら，信号処理という要素技術に関して皆で考えたり検討する場はあまり存在しません．多くの人と知恵やアイデアを闘わせたいと考えている方も多いのではないでしょうか．

そこで編集部では，下記の要領でディジタル信号処理に関する勉強会「ディジタル信号処理技術研究会」を計画いたしました．

＊　　＊

▶ **主催者**：Interface編集部
▶ **事務局担当**：CQ出版(株)　出版局　相原 洋
▶ **参加資格者**：企業(学校を含む)において，研究開発・設計をおもな業務とされている方で，研究会に年4回以上出席でき，積極的な技術交流ができる方
▶ **会の目的**：ディジタル信号処理技術に関する情報の相互交換(製品ノウハウに関することは対象にしません)，技術力向上・啓発のためのセミナー開催，交流・懇親など
▶ **会の運営(案)**
- 研究会の開催：2か月に1回，たとえば偶数月・第一水曜日の午後から3時間ほど
- 開催場所：CQ出版(株)セミナー・ルーム(東京・巣鴨…JR山手線・巣鴨駅から徒歩1分)
- 研究会会報の発行：年4回ほど(将来合意の上)
- 会費：2万3000円/年(運営の実費)
- 会員募集の締切日：2004年5月末日
- 研究会発足：2004年6月初旬予定

＊　　＊

本研究会は，研究・啓発と技術交流を目的とします．あまり多くの方が参加されますと円滑な技術交流が行えない可能性がありますので，メンバは数十名程度に絞らせていただきたいと考えております．

本研究会への加入ご希望の方は，
- お名前と年齢
- お勤め先(社名，住所)，所属部課名，連絡先電話番号
- 担当業務の内容(できるだけ詳しく)

をご記入のうえ，aihara@cqpub.co.jpまで電子メールにてお寄せください．

Interface

次号予告

2004年6月号は
4月24日発売です

最新メカトロ技術と組み込み制御技術の結晶

二足歩行ロボットの制御技術

ロボットのシステム構成と通信技術/ロボット制御回路の設計/CPLD＆FPGAを使ったサーボ・モータ制御回路のロジック設計/市販部品と特注部品によるロボットの機構設計/ロボットを制御するアルゴリズムとプログラミング/ロボットに使われるセンサ技術

二足歩行ロボットは，日本が得意とする最新のメカトロニクス技術や組み込み制御技術，経験と熟練によるモノ作り技術の集大成です．マイコン制御技術(ハード＆ソフト)，センサ技術，モータ制御技術，パワー駆動技術，電源設計技術，通信制御技術など，組み込み技術のほとんどが使われています．さらに機械・機構設計ではCAD/CAMを使って部品を製作しなくてはなりません．

この組み込み技術は，自動車，家電製品，携帯電話，産業用機器，航空機など，あらゆる機器を構成するために必要な要素技術です．

まず，二足歩行ロボットを作っている業界全体の大きな流れ，そこで使われている技術の動向をまとめ、具体例としてロボット試作に使われる技術を紹介します．屈伸，二足歩行，片足立ち，股割り，パフォーマンス演技などC言語によるプログラム・ソースもすべて紹介します．

編集後記

●3月1日から新しい新聞が誕生した．といっても，日本工業新聞だった「FujiSankei Business i」．工業という側面をやわらかくして，経済面や国際面を充実したという感じ．中国株価の情報を毎日載せているのはこの新聞だけらしい．しかし競争相手は新聞だけじゃない．インターネット対応をどうするかだ．（檀）

●食玩でブルース・リーが出てしまった．我が人生に大いなる影響を与えた人なので，思わず手を出してしまったのだが，それがきっかけとなって再びリー様に対する情熱が吹き出しそうになってきているのを必死で抑えている．もう何十年も前の人なので，アイテムを集め始めたら確実に破産するぞ(^^;;)．（＝IO）

●小学生のとき，先生が「世の中，いろいろなものが石油でできている．みんなが着ているジャージもズックも石油だし，最近では肉も石油でできている」と言った．石油製の肉なんてマズそうだけど，ぜひ食べてみたいと思い，十数年間も探し続けている．いまだに見つからないけど，本当にそんなものがあるのかなぁ….（もみ）

●偶然続き．九州の友達との会話に出てくる北海道の人とは私の友達だった．先日，専門用語の意味を米国人に質問したら，彼も前日にその単語の使い方について考えていたそうだ．知り合いの出版記念パーティで友達と鉢合わせ．最大級の偶然は，知り合った米国人(米国在住)は私の友人と知り合いだった．（太陽熱）

●今月号の付録を編集するにあたり，古い資料をひっくり返したり，廃棄寸前の98を引っ張り出してCバス・ボードの動作を確認したりと，もう大変でした．それにしても，PC-9801シリーズって，いろいろあったんですねぇ～(過去形かい)．今でも98を使い続けるファンサイトなどは，もう涙なくしては見られませんヨ(^^;)．（M）

●特集担当だというのに盛大に風邪をひいてしまい，周囲に多大な迷惑をかけることに．薬を飲んでも全然熱が下がらないので，布団を余分に被って電気毛布でガンガンに熱して汗をかくという荒業で対処．汗をダラダラ流したあとはスッキリ平熱になり，風邪も快方に．民間療法でもけっこう効くものですね．（み）

●花粉の時期がきて，天気予報でも毎日の花粉状況が表示されるようになりました．毎年のことだけれど，これから五月上旬までは少しうっとうしい時期です．毎日薬を飲まなければならないし，薬を飲んでいるので眠気にも勝たなければならないし．でも，今年は花粉が少ないとのことなので昨年よりは良いはず….（Y2）

●最近，良い歯医者はないかと友人に聞いて回っているが，なかなか見つからない．良い歯医者と悪い歯医者があると聞くが，見分けがどうにもつかない．小耳に挟んだ情報だと良い歯医者は土曜日まで診察しているところの方が多いらしい．なぜなら，休みが多いところは治療費をふんだんに取っていると聞いた．（七味）

お知らせ

■ 読者の広場

本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

■ 投稿歓迎

本誌に投稿をご希望の方は，連絡先(自宅/勤務先)を明記のうえ，テーマ，内容の概要をレポート用紙1～2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です(送り先はsupportinter@cqpub.co.jpまで)．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

■ 本誌掲載記事についてのご注意

本誌掲載記事には著作権があり，示されている技術には工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は，所有者の許諾が必要です．また，掲載された回路，技術，プログラムなどを利用して生じたトラブルについては，小社ならびに著作権者は責任を負いかねますので，ご了承ください．

本誌掲載記事をCQ出版(株)の承諾なしに，書籍，雑誌，Webといった媒体の形態を問わず，転載，複写することを禁じます．

■ コピー・サービスのご案内

本誌バックナンバーの掲載記事については，在庫(原則として24か月分)のないものに限りコピー・サービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．

●コピー料金(税込み)
1ページにつき100円

●発送手数料(判型に関わらず)
1～10ページ：100円，11～30ページ：200円，31～50ページ：300円，51～100ページ：400円，101ページ以上：600円

●送付金額の算出方法
総ページ数×100円＋発送手数料

●入金方法
現金書留か郵便小為替による郵送

●明記事項
雑誌名，年月号，記事タイトル，開始ページ，総ページ数

●宛て先
〒170-8461　東京都豊島区巣鴨1-14-2
CQ出版株式会社　コピーサービス係
(TEL：03-5395-4211，FAX：03-5395-1642)

■ お問い合わせ先のご案内

●在庫，バックナンバー，年間購読送付先変更に関して
販売部：03-5395-2141

●広告に関して
広告部：03-5395-2133

●雑誌本文に関して
編集部：03-5395-2122

記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface

©CQ出版(株)　2004　振替　00100-7-10665
2004年5月号　第30巻　第5号 通巻第323号)
2004年5月1日発行(毎月1日発行)
定価は裏表紙に表示してあります

発行人／増田久喜
編集人／山形孝雄
編集／大野典宏　村上真紀　山口光樹　大竹友美　小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／森　祐子　唐沢睦子
広告／澤辺　彰　中元正夫　菅原利江

発行所／CQ出版株式会社　〒170-8461　東京都豊島区巣鴨1-14-2
電話／編集部(03)5395-2122　FAX／(03)5395-2127
広告部(03)5395-2133　URL http://www.cqpub.co.jp/interface/
販売部(03)5395-2141　E-mail supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1-14-2 Sugamo, Toshima-ku, Tokyo 170-8461, Japan
印刷／クニメディア株式会社　美和印刷株式会社
製本／星野製本株式会社

日本ABC協会加盟誌
(新聞雑誌部数公査機構)　ISSN0387-9569